HITACHI
MB-6890
LEVEL-3
DISK BASIC
MANUAL

V 1.1

CAUTION

1. Reproduction of a part of whole of this manual without our approval is prohibited.

2. Contents of this manual are subject to change without advance notice.

3. We would appreciate receiving comments or questions if there are any points that are not clear, erroneous or whatever other points you may have.

4. Irrelevant to Item 3 of the above, Hitachi is not to be held responsible for operation results.

CONTENTS

1. HANDLING

1.1 EXPLANATION OF MANUAL

This manual describes the HITACHI MB-6890 LEVEL-3 DISK BASIC (hereinafter abbreviated as DISK BASIC). It consists of Chapters 1, 2, 3 and 4. The following is a guide in using this manual.

CHAPTER 1 HANDLING

Explanation is given on how to start the DISK BASIC by connecting MB-6890 and the MP-3540 Minifloppy Disk, and regarding utility programs of the system diskette.

CHAPTER 2 LANGUAGE SPECIFICATIONS

Commands, statements, functions, and features of the system variables of DISK BASIC are described.

2.1 OUTLINE OF BASIC: Description of fundamental points of BASIC

Features possessed by LEVEL-3, such as characters, array, expressions and operations, screen editor, file distributor, graphic mode, etc., that can be used in DISK BASIC are described.

2.2 INSTRUCTIONS (COMMANDS AND STATEMENTS)

Commands and statements related to disk operation of DISK BASIC are listed in the slphabetic order, and their features and formats are described.

2.3 FUNCTIONS AND SYSTEM VARIABLES

Functions and system variables related to operation of DISK BASIC are described.

PURPOSE : Refers to features of instructions and functions and meaning of system variable.

FORMAT : Shows formats and sequence of parameters in instructions and functions. The symbols used here have the following meaning:

1. Input the items shown with capital letters as is.
2. Users are to specify the items placed between ‹and› .
3. Items placed between [] are optional and may be omitted. If they are omitted, default values or values specified before are applied by the instructions or functions.
4. Items having continuous ellipses may be repeated for any number of times within a line. For example, A, B, X, Y, etc. in this case of ‹variable› [, ‹variable› ....]
5. If written like ⎧ ⎫ or ⎧ ⎫ , either the top or ⎩ ⎭ ⎩ ⎭ bottom may be used. In this case, either a semicolon or comma may be used.
6. All symbols including a comma and default period other than those in the above explanation are grammatically needed, and must be written in programs unless they can be omitted singly (for example, a comma placed between brackets, [,]. (Note that if shown as [,R], the comma may be omitted if R is omitted, but if R is written, the comma must also be written.)
7. ⌜ ⌟ are used as quotation marks in the explanatory sentences. Therefore, ⌜ " ⌟ means the symbol in the program.

EXPLANATION : Concrete actions and values of instructions and functions and cautionary remarks on use are described.

CHAPTER 3 DISK OPERATION

How to operation the disk files and how to create commands and programs are concretely explained using examples of address catalog programs. Reading of this chapter is recommended to beginners. The description is made assuming a certain degree of mastering of the HITACHI MB 6890 Level-3 BASIC (hereinafter abbreviated as L-3 BASIC), but on parts related to the disk, the description is given in an easy way to ensure beginners understanding.

CHAPTER 4 DATA

Character code tables, error messages, memory maps and dump lists are given in this chapter.

Prior to start operating DISK BASIC after reading this manual, the users are required to fully understand the L-3 BASIC specifications and connecting method of MP-3540 and MP 6890 by reading the L-3 BASIC manual and the Minifloppy disk MP-3540 manual. We recommend that operators who use DISK BASIC for the first time read this manual in the order of Chapters 1, 3 and 2.

Description format of this manual is different from the L-3 BASIC manual in the following points:

1. In Chapter 2, purposes only are described on instructions and functions not related to the disk. Therefore, for further details of these, refer to the L-3 BASIC manual.

   On instruction and functions that are not related to the disk but bear partial alteration, explanation is collectively given in "2.1.18 Different Points with LEVEL-3 BASIC".

2. In "2.1 OUTLINE OF BASIC", fundamental items are described although they may be overlapping with the L-3 BASIC Manual.

3. Words having a prefix, like mini before floppy, are written in one word.

4. 「 」 are used as quotation marks in explanatory sentences, discrimination from the 「"」 as a syntax.

5. Consideration is paid to be consistent with the L-3 BASIC Manual, but whatever characters or terms that seem to be not appropriate are and will be improved.

1.2 ACTIVATION OF LEVEL-3 DISK BASIC

Connect MB-6890 and MP-3540 and turn on the power switch of MP-3540 after thorough reading of "1.2 CAUTION ON USE" through "1.4 CONNECTION METHOD" of the L-3 BASIC Manual and the MP-3540 Minifloppy Disk Manual. Then, insert the diskette associated with this manual to Drive 0 following the MP-3540 Manual description and turn on the MB-6890 power switch. The following information is displayed on the screen. (This applies to the case where the MB-6890 MODE switch is in the state of ▙ 0, that is, the switch button is in lifted state.)

```
HITACHI LEVEL-3 DISK BASIC Version 1.0
Copyright (c) 1980 By Microsoft
17944 Bytes Free
Ready
-
```

The DISK BASIC is in operable state on MB-6890. The figure of 17944 shown on the third line of the screen indicates the number of memory bytes that can be used for the program. MB-6890 is provided with 4 types of display mode, in the same way as L-3 BASIC, and number of memory bytes that can be used in each mode of the 4 types varies as shown in the following table.

Display Mode	Usable number of memory bytes
1 40-character normal mode	25112
2 80-character normal mode	24088
3 40-character high resolution mode	17944
4 80-character resolution mode	9752

The information in the above display indicates that the MB-6890 is in the 40-character high resolution mode. For details of how to set and use these display modes, refer to the L-3 BASIC Manual.

The following factors are conceivable if the display shown in the above is not obtained correctly. Check these points.

1. The MB-6890 or MP-3540 power cord is not plugged into the power outlet correctly.
2. No I/O card MP-1800 is mounted to the extension slot of MB-6890.
3. The MP-3540 connector is not correctly connected to I/F-1 of the panel for interface extension.
4. The system diskette is not inserted to MP-3540.
5. The MP-3540 motor lamp is not lighting, or the diskette is not fully inserted. (Re-insert it after reading the MP-3540 Manual again.)
6. Procedures of MP-3540 power switch on, diskette insertion and MB-6890 power switch on were not performed in the correct sequence. (Re-read the MP-3540 manual).
7. The number of usable memory bytes varies by the condition of the MB-6890 internal DIP switch and mode switch in the front and setting of the CONFIG program.

When the number of horizontal display characters is expressed as W, resolution mode as R (0: high resolution, 1: normal), the maximum drive number set by the CONFIG program (see Clause 1.3) as D, and number of FCB as F, B which is the number of usable memory bytes can be calculated in the following formula:

$$B = 26866 + \frac{128}{5} \times W \times (7 \times R - 8) - 142 \times F - 162 \times D$$

1.3 UTILITY

In the system diskette associated with this manual, the following 4 utility programs are stored in addition to the DISK BASIC codes. Purposes and use methods of these programs are described in this clause.

FORMAT
SKIP 2
CONFIG
DISKCOPY

The 3 programs of FORMAT, CONFIG and DISKCOPY are written in BASIC but SKIP2 is a program written in a machine language. It is a program to be called out during FORMAT program execution and not to be executed singly.

1.3.1 Format (Diskette initialization)

Diskettes available in the market cannot be used for read/write of data and programs if they are mounted in the disk unit. New diskettes are nothing but plain disks and they can be used in read and write operations in the disk unit only when certain information is written on the diskette tracks (concentric memory area similar to disk record grooves). Initialization of the diskettes is called formatting.

The following two procedures are needed for formatting of brand new diskettes:

Step 1: Write specified data to all tracks of a new diskette using the FORMAT program stored in system diskette. (Beware that if diskettes having data and programs already recorded are used, the written data and programs are erased.)

Step 2: Initialize the directory track (see Item 2.1.17) using the DSKINI instruction. The diskette can now be used for BASIC instructions and functions.

When there is a diskette that contains data and programs but cannot be used for read/write since the recorded data was destroyed for some reasons, execute the DSKINI instruction outlined in Step 2. If this is done, the data and programs recorded on the diskette can no longer be read by functions or instructions other than DSKI$.

Actual formatting procedures

Step 1

(1) Insert the system diskette associated with this manual to Drive 0 and input as follows:

LOAD "FORMAT" [RETURN]

This loads the FORMAT program stored in the system diskette. Execute the program using the RUN command.

(2) There is a display of 「Which Drive to FORMAT (1, 2, or 3)?」 on the screen. Check the diskette to format and insert it to the drive, and input the drive number and press the [RETURN] key.

Make sure that the diskette to format is not a diskette that contains important data or programs.

The message of 「Are you sure?」 is displayed on the screen. Check that the drive number was correct then input Y [RETURN].

(3) Input N [RETURN] to terminate formatting. Then the message of 「FORMAT program aborting」 is displayed and the system returns to command mode.

(4) When Y [RETURN] is input, the ⌜SKIP2⌟ machine language program is automatically loaded, the drive acts for 2 to 3 minutes for the formatting work, and the message of ⌜FORMAT finished⌟ is displayed on the screen, terminating the program.

Other diskettes can be formatted in succession using the RUN command.

Step 2

(1) Check that the system diskette is not inserted in the drive.

(2) Insert the diskette that was formatted in Step 1, or a diskette that contains unneeded data and programs, to Drive 0 or 1.

(3) Input either one of the following:

DSKINI 0 [RETURN] (when the diskette is inserted in Drive 0)

DSKINI 1 [RETURN] (when the diskette is inserted in Drive 1)

There will be the following message on the screen:

⌜Are you sure (Y or N) ?⌟

Input Y if execution is agreeable, otherwise N.

If Y is input, the directory track is initialized. The system diskette associated with this manual needs no initialization of this type since it has been given all the necessary treatment. Be careful that if the FORMAT program or DSKINI instruction is executed, all data that has been written in will be lost, and the DISK BASIC can no longer be activated.

When the FORMAT program is executed, 128 pieces of a certain value is written in all sectors. Therefore, if

the execution is applied to the system diskette, all the BASIC codes are destroyed. Also, if the DSKINI instruction is executed to the system diskette, the directory track is initialized, and though the BASIC codes are not destroyed, the system can no longer recognize existence of the BASIC codes. Accordingly, the BASIC codes are destroyed when some output (SAVE, PRINT#, PUT, OPEN "O", or OPEN "R") is given to the disk.

Program list

```
10 DEFINT A
20 DIM A (1000)
30 INPUT"Which Drive to FORMAT(1, 2, or 3) ";DRV
40 IF DRV<1 OR DRV>3 THEN 30
50 INPUT"Are you sure";A$
60 IF A$<>"Y" THEN 170
70 ADR=0
80 X=0
90 ADR=VARPTR(A(0))
100 LOADM"SKIP2",ADR-&H4000
110 POKE ADR+2,DRV
120 DEFUSR0=ADR
130 X=USR0(X)
140 POKE &HFF20,0
150 IF PEEK(ADR+3)=0 THEN 180
160 PRINT"Error occurred during FORMAT"
170 PRINT"FORMAT program aborting":END
180 PRINT"FORMAT finished":END
```

Execution example

```
LOAD"FORMAT"

Ready
RUN
WHich Drive to FORMAT(1, 2, or 3)? 1
Are you sure? Y
FORMAT finished
```

```
Ready
DSKINI 1
Are you sure (Y or N)? Y
Ready
```

1.3.2 SKIP 2

This program is written in a machine language, and is used by the FORMAT program after being automatically called from the disk. Therefore, for execution of the FORMAT program, a diskette containing the SKIP2 program needs to be inserted in Drive 0.

DISKCOPY, one of the utilities, may be used for backup of this program, but there is a method to record the program in another diskette, in procedures as described in the following:

(1) Check that the system diskette is inserted in Drive 0 and input as follows:

```
LOADM "SKIP2"
```

(2) Then enter the program stored in the memory to another diskette. Insert the diskette to which the program is to be entered to Drive 1, and key in as follows:

```
SAVEM  "1 : SKIP2", &H4000, &H41FF, &H4000
```

This completes the operation. When the program is entered in the memory using the LOAD Instruction, the location is in the range of &H4000 through &H41FF. However, it is entered in the range of &H1C66 through &H1E65 within the FORMAT program, and this should be kept in mind.

Also beware that if the file name or the head address (the first &H4000) is changed, the FORMAT program does not operate normally.

Execution example

```
LOADM"SKIP2"

Ready
SAVEM"1:SKIP2",&H4000,&H41FF,&H4000
```

1.3.3 CONFIG (Setting of maximum Drive NO. and number of FCB)

This program designates the number of usable drives and the number of files that can be opened concurrently.

When the RUN command is executed, first, ⌜Highest Drive Number (0-3)?⌟ is displayed. Input the maximum number of usable drives and press the [RETURN] key.

Then, ⌜Highest FCB Number (0-15?⌟ is displayed. Designated a value of number of files that can be concurrently opened less one.

The reason why one must be subtracted is that the objective of the designation is FCB (File Control Block), and one FCB is needed for I/O operation to the disk by opening a file. However, since the facility that is equivalent to one FCB is contained in the system, when FCB of 'n' pieces is preserved, 'n+1' files can be opened. Accordingly, the value to be set in this program is the number of files to be concurrently opened less one.

Another thing to be noted is that the maximum number of drives designated does not necessarily restrict actual use of drive numbers exceeding the maximum. If an interface and drive itself are available, common processing like SAVE or OPEN is possible. 4 instructions; DSKO$, DSKI$, DSKINI and DSKF, and functions cannot be used.

This program function re-writes the BASIC codes of the disk only and does not affect the DISK BASIC that has been stored in the memory and being activated. To make the values designated by this program valid, either turn on

the power again or restart using the NEW ON command.

In the system diskette, the maximum drive number is 1 and the number of FCBs is 4.

The number of bytes in the user RAM area that can be used changes according to the value set by this program. For further details, refer to 1.2 ACTIVATION OF LEVEL 3 DISK BASIC.

Program list

```
10  INPUT"Highest Drive Number(0 - 3)";D
20  IF D <0 OR D> 3 THEN 10
30  INPUT"Highest FCB Number(0 - 15)";F
40  IF F <0 OR F> 15 THEN 30
50  A$=DSKI$(0,1,6)
60  MID$(A$,91,2)=CHR$(D)+CHR$(F)
70  DSKO$ 0,1,6,A$
80  END
```

Execution example

```
LOAD "CONFIG",R

Highest Drive Number(0 - 3)? 1
Highest FCB NUmber(0 - 15)? 4
```

1.3.4 DISKCOPY (Copying of diskette)

This program is for copying of complete contents of one diskette to another diskette, and contents of the diskette that are copied to, are completely erased. Pay particular attention that the diskette is utterly unneeded and make sure that no error is made in inputting the drive number. When the RUN command is executed, ┌Source Drive, Dest. Drive?┘ is displayed. Input the drive numbers of the diskettes to which the contents are to be copied to, and from. Input the both numbers by delimiting them with a comma, (,), and then

press the [RETURN] key.

As the copying progresses, the track number in the process is displayed, and the copying work completes in about 3 minutes.

Since this program copies all tracks, backup of the system diskette can be made. A backup diskette should be made as soon as possible since the DISK BASIC cannot be used once the system diskette is destroyed.

Program list

```
10 CLEAR 3000
20 DIM S$(16)
30 INPUT"Source Drive, Dest. Drive";SD,DD
40 FOR T=0 TO 39
50 PRINT"track = ";T
60 FOR S=1 TO 16
70 S$(S)=DSKI$(SD,T,S)
80 NEXT S
90 FOR S=1 TO 16
100 DSKO$ DD,T,S,S$(S)
110 NEXT S
120 FOR S=1 TO 16
130 IF S$(S)<> DSKI$(DD,T,S) THEN PRINT"BAD ";T;S
140 NEXT S
150 NEXT T
160 PRINT "Copy Complete"
170 CLEAR 300
180 END
```

Execution example

```
RUN "DISKCOPY"
Source Drive, Dest. Drive? 0,1
track = 0
track = 1
track = 2
track = 3
track = 4
track = 5
    :
    :
    :
```

2. LANGUAGE SPECIFICATIONS

2.1 OUTLINE OF DISK BASIC

2.1.1 Operation Mode

When the system starts to operate and the BASIC is activated, the message of ⌜Ready⌟ is output, indicating that the BASIC is in the command level, that is, waiting for a command. As this time, the BASIC receives and executes all types of commands.

When a command or statement of the BASIC is input without being assigned with the line number, it is immediately executed. This is called execution in the direct mode. The mode is very useful in using the BASIC as a calculator for arithmetic operations.

Commands and statements that are input with the line numbers are stored in the memory as programs. The stored programs can be executed by the RUN command, and GOTO or GOSUB statement.

2.1.2 Program

A program consists of lines where statements of the BASIC are described. They are BASIC commands and statements. The format of each BASIC line is as shown below:

NNNN <BASIC statement> [: <BASIC statement> ...] RETURN

NNNN is the number of the line. The line number indicates the sequence of storing the lines in the memory. Integers in the range of 0 through 63999 are used for the line numbers.

Two or more BASIC statements can be written in a line, and this is called multi-statements. These statements must be delimited with a colon ⌜:⌟ . End of input is realized by pressing the RETURN key. Up to 255 characters including

the line number can be input in a line.

The BASIC executes the input program in the sequence of the line numbers and destinations to branch to, are referred to by the line number. The line numbers are used as markings to show the objective lines when the users edit programs. When editing, period, ".", may be used to indicate the line that the BASIC is currently recognizing during execution of the LIST, EDIT, or DELETE command.

2.1.3 Usable Characters

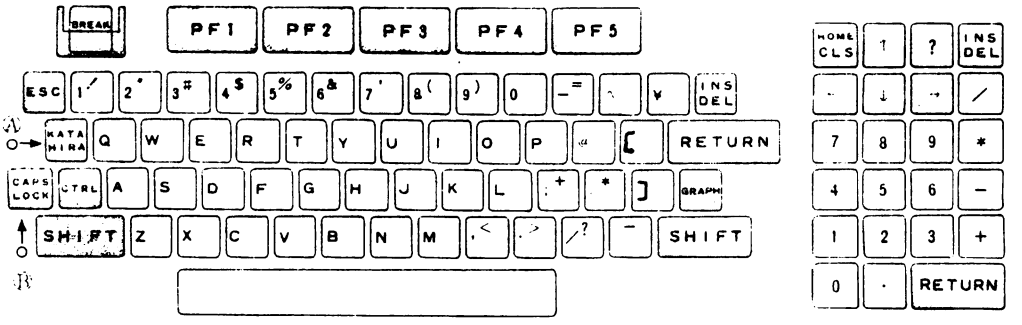
Alphabetics, Kata-kana, Hira-gana, numeric characters, symbols and special symbols can be used by the DISK BASIC. Both capital and small characters of alphabetics can be used. Numeric characters from 0 through 9 can be used.

Hira-gana can be used in the DISK BASIC. Also, several special symbols are provided.

For details fo the characters, refer to the character code tables in Chapter 4.

2.1.4 Keyboard

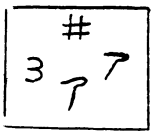
Layout of keys on the LEVEL-3 Keyboard is shown below.



The keyboard layout is conformity to JIS (Japan Industrial Standards), and both alphamerics and kana characters can be input. At the right side, keys for cursor control and ten keys for numeric input are located. Programmable

function keys that can be freely defined by users as located at the top.

Use the KATA HIRA key located at the left corner to key in alphabetics, Kata-kana and Hira-gana from this keyboard. When the key is pressed, the lamp at its left changes to either red or green. When the lamp is red, the Kata-kana character marked on the key top can be input. When the lamp is green, Hira-gana can be input. When the lamp is not being lit the alphameric can be input. When the lamp is green, Hira-gana can be input. When the lamp is not being lit the alphameric can be input. To input alphabetics or small kana characters or symbols, use the SHIFT key. When the character key is pressed while the SHIFT key is being pressed, small characters are input. A diagram of the key top marking and procedures to input an objective character are shown in the following.



3: Press the key as is.

#: Press the objective key while pressing the SHIFT key.

: Press the KATA HIRA key to turn the left lamp to red and then press the objective key.

: Press the KATA HIRA key to turn the left lamp red, and press the objective key while pressing down the SHIFT key.

: Press the KATA HIRA key to turn the left lamp to green, and then press the objective key.

: Press the KATA HIRA key to turn the left lamp to green, and then press the objective key while pressing the SHIFT key.

NOTE: The lamp color at the left of the [KANA HIRA] key changes only when the key is pressed. Therefore, when inputting Kata-kana or Hira-gana continuously, the [KATA HIRA] key need not be pressed each time of the character input.

In the ordinary state, alphabetics are input in capital letters. To input small letters in normal state as in the case of ordinary typewriters, press the [CAPS LOCK] key located at the left corner. When the key is pressed, the lamp below it lights up to indicate that the keyboard is in the state of small character input. To release the state, just press the key again. The lamp goes out and small character input state is released.

A click sound is heard when a key is pressed and this confirms that the key is pressed. The volume of the click sound is adjustable by the knob located in the front panel. When one key is kept pressed for a certain length of time, the same character is automatically repeated.

Use the [GRAPH] key to input graphic symbols. When the objective key is pressed while the GRAPH key is being pressed, the graphic symbol of the key is input.

The cursor control keys located at the top of the ten keys are used when editing programs on the screen. For details of how to use these keys, refer to 2.1.11 Screen Editor.

A character string is defined to each of the programmable function keys located at the top part of the keyboard, so that actions of pressing a number of input keys can be substituted by pressing one of these function keys. These function keys react to the [SHIFT] key, and this makes 10 different type uses of the function keys. For details how to use these function keys, refer to the L-3 BASIC Manual.

The [BREAK] key, combined use of the [CTRL] and [C] [CTRL] + [S] and [CTRL] + [D], and the reset switch in the front panel are provided as keys to hold BASIC operations. The [BREAK] key holds BASIC to execute the current program and sets the system to the command level.

[CTRL] + [C] performs the same thing.
[CTRL] + [S] temporarily stops execution of the LIST instruction. When the LIST instruction is executed, if the number of program lines to be displayed is greater than the screen size, the screen is scrolled as the latter lines are displayed. If a specific line must be read carefully, press these keys when the particular line appears in the screen. The scroll stops. (Scroll restarts when another key is pressed.) When key-input of either BREAK or [CTRL] + [C] is made, execution of the LIST instruction is held and BASIC enters the state of waiting for input.

Key-input of [CTRL] + [D] functions similar to that of [BREAK], except holding of the operation is stronger. The reset switch performs the strongest holding. However, beware that holding of operation by the reset switch during execution of SAVE and SAVEM commands may ruin the program being recorded at the time or programs and data in the diskette. Use the combination of [CTRL] + [D] instead of the reset switch.

2.1.5 Constants

Constants are the values that BASIC uses during operation. There are two types of constants; character constants and numeric constants.

(1) Character constants

Character constants are strings consisting of up to 255 alphabetics, numeric characters, Kata-kana or Hira-gana characters placed between quotation marks ("). Character of 0 length is called "Null string".

Examples: "ABC"
"トウキョウ " (Kata-kana for Tokyo)
"123"
"ひらがな" (Hira-gana for hira-gana)
" " (Null string)

(2) Numeric constants

Numeric constants are positive or negative (including 0) numbers. There are the following three types in numeric constants:

a) Integer constants

Integers in the range of - 32768 through 32767

Decimal constants

Decimal constants are expressed with numeric characters of 0 through 9 and + and -. 「%」 may be used to identify that it is an integer.

Hexadecimal constants

Hexadecimal constants are expressed with numeric characters of 0 through 9 and alphabetics of A through F and + and -. 「&H」 is placed at the head to indicate that it is a hexadecimal constant.

Octal constants

Octal constants are expressed with numeric characters of 0 through 7 and + and -. 「&」 or 「&O」 is placed at the head to indicate that is an octal constant.

Examples; 123
256%
&H2F
&0177
&123

b) Single precision constants

Single precision constants are numbers consisting of up to 7 significant digits and in the range of about 3.4×10^{-39} through about 1.7×10^{38}. [!] may be written after the number to indicate that it is a single precision constant. In the case of single precision, constants are stored in precision of 7 digits and are displayed up to 6 digits.

Fixed point type constants

Fixed point type single precision constants are positive or negative numbers (including 0), and decimal point is included in the number.

Examples: 1.234
-6.54321
3.0!

Floating point type constants

Floating point type single precision constants are positive or negative numerics expressed in the exponent form. These constants are expressed in the form of a signed integer ([+] may be omitted) or a numeric of fixed point type (mantisa part) followed by the [E] character and a signed integer ([+] may be omitted) of the exponent part.

Examples: 1.234E2
0.123456E36

c) Double precision constants

Double precision constants are numbers consisting of 8 or more significant digits and in the range of about 3.4×10^{-39} through about 1.7×10^{38}. [#] is written after the number when the number is not easily distinguished from single precision constants.

In the case of double precision, constants are stored in precision of 16 digits and displayed in 16 digits.

Fixed point type constants

The expression method is the same as that of single precision, but write ⌜#⌟ after the number if the significant digits are no greater than 7.

Examples: 1.2345678

1.23#

Floating point type constants

The expression method is the same as that of single precision, but use ⌜D⌟ instead of ⌜E⌟ of the exponent part.

Example: 1.23D10

2.1.6 Variables

Variables are assigned "variable names" and used to represent values in a BASIC program.

(1) Variable names

Variable names are subjected to the following restrictions:

a) Use an alphabetic at the head and follow it with alphabetics or numeric characters. Up to 255 characters may be used.

b) BASIC defines variable names with the first 16 characters plus attribute characters (described later).

c) No BASIC reserve word (keyword of commands, statements, delimit, etc.) should be used at the head.

Examples: ABC, IDATA1 ..... Permitted.
1DATA, DATA1, TOWARD ..... Not permitted.

(2) Character type variables

Write the $ attribute character at the end of variable names. Up to 255 characters can be written in characters type variables.

Examples: DISK$
A$

(3) Integer type variables

Write the % attribute character at the end of variable names. Each variable of this type requires a 2-byte memory space to be stored.

Examples: IDATA%
ISUM%

(4) Single precision type variables

Write the ! attribute character at the end of variable names. If the attribute character is omitted, a single precision type variable is automatically assumed, but if the type declaration has been made the declared type is automatically assumed. Each variable of this type requires a 4-byte memory space to be stored.

Example: TEN
SUUCHI!

(5) Double precision type variables

Write the # attribute character at the end of variable names. Each variable of this type requires a 4-byte memory space to be stored

Examples KEISAN
YOSAN

2.1.7 Array

Array is a collective name of a group of values of tables to which reference is made with the same variable name, and the value in the array is called an element. Each of these elements is referred to with an array variable accompanied with subscripts of integer expression. All array variable names have subscripts in the same number as the dimensionality of the array. For example, M (10) is treated as a value of an one-dimensional array and X (1,4) as a value of two-dimensional array. The same applies to multiple dimensions. 4 or greater dimension arrays are possible though it depends on operation mode (user area size), number of array variables and number of elements.

Examples: A (10)
B (1, 2)
C (10,10,10)
D$ (10,80)

2.1.8 Conversion

DISK BASIC converts types of numeric constants and numeric variables to others as needed.

(1) When a numeric constant or variable of a type is assigned to a numeric variable of other type, the value is stored with its original type being converted to the type that has been declared by the variable name. If a character is assigned to a numeric, or vice versa, it causes an error.

Example: A% = 23.45
↑—23 is stored.

(2) When a real type numeric constant or variable is assigned to an integer type, the fraction portion is stored as an integer by counting fractions of .5 and over as a whole number and disregarding the rest. If the rounded value is not in the range of -32768 through 32767, it causes an error.

```
Example:  B% = 12.56
          ↑
          └─13 is stored
```

(3) When double precision type is converted to single precision type, rounded value of the first 7 digits is stored.

```
Example:  C = 3.141592653589793
          ↑
          └── 3.14159 is stored.
```

(4) When single precision type is converted to double precision type, contrary to the above case, a value arrived at by rounding the 7th digit is effective. This makes the absolute value of the error between the original single precision value and converted double precision value no greater than 6.3E-8. The reason is in numerics of single precision, precision obtainable is limited to up to 7 digits.

```
Example:  10 A=2.04
          20 B# =A
          30 PRINT A,B#
          RUN
           2.04           2.039999961853027
```

(5) Therefore, when assigning a constant of not exceeding 7 digits to a double precision variable, the error can be made smaller by defining it as a double precision constant with the # symbol written at the numeric end.

```
Example:  10 A# =2.04 #
```

(6) In arithmetic operations of expressions, the calculation is made matching the item having the highest precision in the right side, and when assigning the result to the left side, conversion procedures between expressions are followed through.

```
Example:  10 D# =6# /7
          20 E# =6/7
          30 PRINT D#, E #
          40 END
          RUN
          .8571428571428571
          .8571428656578064
```

(7) In logical operations, each item is rounded by counting fractions of .5 and over as a whole number and disregarding the rest. Logical operation is executed only after this conversion to an integer. When the converted value is not in the range of -32768 through 32767, it causes an error.

```
Example:  10 PRINT 7.35 OR 16
          RUN
          23
```

2.1.9 Expressions and Operations

The term expression means a group of constants and variables combined by operators for the simple purpose of obtaining a constant or variable or a certain value of characters or numerics.

```
Examples:  3.1415926
           "BASIC"
           H$
           10-8
           A + B
           X<Y-Z
```

Operations associated with DISK BASIC are given with the following priority sequence:

1. Expression placed in parentheses
2. Function
3. Arithmetic operator
4. Relational operator
5. Logical operator

(1) Arithmetic operator

a) Operation sequence

1. Power multiplication (Λ)
2. Minus symbol (-)
3. Multiplication (*) and division (/) of real numbers
4. Division (¥) of integers
5. Remainder (MOD)
6. Addition (+) and Subtraction (-)

Use parentheses to change the operation sequence. Also, parentheses may be used to make the operation sequence clear if it is not clear.

Correspondence between ordinary expression and BASIC expression is shown below.

Ordinary expression	BASIC expression
2X+Y	2*X+Y
YxY÷2	Y*Y/2
X^2+2X	XΛ2+2*X
$(X^2)Y$	(X 2)Λ Y or X 2 Y
$X^{(Y2)}$	XΛ (Y 2)

b) Division of integers and remainders

Division of integers is expressed by the YEN symbol (¥). When the divisor and dividend are real numbers, their fraction portion, if any, is rounded to an

integer by counting fractions of .5 and over as a whole number and disregarding the rest, prior to the division. The quotients are rounded to integers.

Example: 26.68¥ 6. 99=27¥ 7=3

Remainders are calculated by MOD. Remainders from division of integers are obtained from the MOD operation.

Example: 26.68MOD6.99=27MOD7=6

(2) Relational operator

Relational operators are used to compare two values. If the comparison result is true, -1 is given and if not true, 0 is given. The result can be used to change flow of the program using the IF-THEN-ELSE statement.

Operator	Relation	Marking
=	Equal	X=Y
<> ,><	Not equal	X<>Y
<	Smaller	X<Y
>	Greater	X>Y
<=, =<	Equal or smaller	X<=Y
>=, =>	Equal or greater	X>=Y

(3) Logical operator

Logical operation functions are provided for bit operations, pool operations and to check various relations. Functions of each operator are listed in the order of higher priority in the following. In the list, 1 means 「true」 and 0 「untrue」 .

① NOT (Negation)

X	NOT X
1	0
0	1

② AND (Logical product)

X	Y	X AND Y
1	1	1
1	0	0
0	1	0
0	0	0

③ OR (Logical sum)

X	Y	X OR Y
1	1	1
1	0	1
0	1	1
0	0	0

④ XOR (Exclusive OR)

X	Y	X XOR Y
1	1	0
1	0	1
0	1	1
0	0	0

⑤ IMP (Inclusion)

X	Y	X IMP Y
1	1	1
1	0	0
0	1	1
0	0	1

6 EQV (Equivalence)

X	Y	X EQV Y
1	1	1
1	0	0
0	1	0
0	0	1

The same as relational operators that can be used for determining program flow, logical operators can be used to combine relation of two or more items for obtaining a value of true or untrue for judgement. If the result is 0, the value is untrue and if other than 0, the value is true.

Example: IF 0<A AND A<100 THEN 80
IF X<0 OR K<0 THEN 100
IF NOT EOF (1) THEN 50

Supplement

With logical operators, items are converted to integers (16 bits) prior to operation. The operation is performed by corresponding each bit, that is, each resulting bit depends on the situation of corresponding bits. Some examples are shown in the following:

-1 OR 0=-1

$-1=(1111\ 1111\ 1111\ 1111)_2$

Therefore, -1 OR 0=-1

23 AND 7=7

$23=(10111)_2$

$7=(00111)_2$

Therefore, 23 AND 7=7

16 XOR 6=22

$16=(10000)_2$

$16=(00110)_2$

Therefore, 16 XOR $6=(10110)_2=22$

12 IMP 5=-9

12=(0000 0000 0000 1100)$_2$

5=(0000 0000 0000 0101)$_2$

Therefore, 12 IMP 5=(1111 1111 1111 0111)$_2$=-9

16 EQV 6=-23

16=(0000 0000 0001 0000)$_2$

6=(0000 0000 0000 0110)$_2$

Therefore, 16EQV 6=(1111 1111 1110 1001)$_2$=-23

There is a relation of X EQV Y=NOT (X XOR Y).

Also there is a relation of NOT X=-(X+1).

2.1.10 Operation of Character Strings

Character strings can be combined by the + operator. Also, they can be compared using the same comparison operators as are used in numerics.

Examples:
```
10 A$="DEFG":B$="HIJKL"
20 PRINT A$+B$
30 PRINT "ABC"+A$+B$
RUN
DEFGHIJKL
ABCDEFGHIJKL

Ready
```

In character string comparison, the character codes for one character each of the character strings are compared. If all the character codes are equal, the two character strings are equal. If the character codes are different, the character string having the smaller code is considered to be smaller. If a character string of one side terminates in the midst of comparison, the shorter character string is considered to be smaller. Blanks before and after each character string have meaning, as shown in the following:

```
Examples  "BBC" < "BC"
          "HITACHI" = "HITACHI"
          "Z&" > "Z  "
          "BASIC" > "BASIC"
          "new" > "NEW"
          "SKIP" < "SKIPF"
           B$ < "9/11/80"          (When B$="8/11/80")
```

Character string comparison can be used to check values of character strings or to place character strings in the alphabetic order.

2.1.11 Screen Editor

(1) Role of RETURN key

Characters and symbols input from the keyboard are not immediately input to the BASIC. Characters input from the keyboard are edited on the screen by the screen editor first, input into the BASIC only when the RETURN key is pressed and then interpreted. What the RETURN key is pressed and then interpreted. What is input is the line where the cursor resides when the RETURN key is pressed. The BASIC checks strings of characters that are input, and if a line number is detected, the line is determined to be a BASIC program and stored in the memory. Otherwise, the line is determined to be a direct command and immediate execution is attempted.

What has to be noted is that the screen resulting from input and editing of the screen editor does not become effective unless the RETURN key is input.

(2) Cursor shift

When an ordinary character or symbol is input, the cursor automatically moves to the next display position. However, when editing a program list or direct command

that is displayed on the screen, the cursor must be shifted to the required positions. LEVEL-3 is provided with the following keys for efficient cursor position shift.

Pressing of these keys moves the cursor toward the arrow direction by one character position.

SHIFT + HOME CLS or CTRL + K

These key operations place the cursor at the home position (top left corner of the screen).

SHIFT + → or CTRL + F

These key operations shift the cursor to the head character of the item that follows the next blank or operator. The cursor can be moved quickly for each keyword or parameter by using this facility.

SHIFT + ← or CTRL + B

These key operations shift the cursor to the last character of the item that follows the blank or operator to the preceding one.

(3)Screen erase

HOME CLS or CTRL + L

These key operations erase the entire screen display and move the cursor to the home position.

CTRL + E

These key operations erase all the display at the right side of the cursor to the end of the line.

The cursor does not move.

CTRL + Z

These key operations erase all display from the current cursor position to the last line of the screen. The cursor does not move.

(4) Insertion and deletion

[SHIFT] + [INS DEL] or [CTRL] + [R]

These key operations do not move the cursor but insert a blank at the cursor right. A new character can be put in the created blank position.

[INS DEL] or [CTRL] + [H]

These key operations delete the character at the left side of the cursor. The cursor moves toward the left side of the cursor. The cursor moves toward left.

(5) Horizontal tab function

[CTRL] + [I]

The cursor moves to the next tab position. Display contents to the part where the cursor moved are erased.

LEVEL-3 is provided with the same horizontal tab function as the typewriters have. The cursor can be stopped at any position as many times as required in the line. When the BASIC is started, 9 tab positions are set, on every 8th digit with the left end being the 0 position.

[CTRL] + [T]

These key operations set the tab position. Place the cursor at the position where a tab position is to be set and press the CTRL and T keys. Tab positions that were set previously are not released by this operation.

[CTRL] + [Y]

These key operation release tab position that has been set. Place the cursor at the position where a tab position is to be released, which is the same procedure as the [CTRL] + [T] Operation, and press the [CTRL] and [Y] keys. This operation does not affect tab positions set to other digits.

2.1.12 Error Message

An error message is displayed when the DISK BASIC detects an error that causes execution stop.

Format of error messages to direct commands is:

```
XXX
```

Format of error messages to program statements is:

```
XXX in lllll
```

XXX is the error message and lllll is the line number where the error is detected. For details of the DISK BASIC error codes and error messages, refer to Chapter 4.

2.1.13 File Descriptor

In LEVEL-3 BASIC, all input/output devices are treated with the concept that they are a file. File is accumulation of information that has meaning. There are two file types in the BASIC; one being program file and the other being data file. Files are defined by file descriptors. File descriptors are normally indentified with " (double quotes) placed at the head and end of the descriptor and characters like character variables or character expressions can be used.

(1) File number and Channel number

In the same way as in L-3 BASIC, in DISK BASIC, input and output are operated by designating files and

channels. Since the operation of external input/output devices is slower than CPU speed, information is temporarily stored in an area called buffer for efficient access. The number unique to the buffer is called file number or channel number, and the two are of the same concept. File numbers are assigned to memory units and channel numbers are assigned to communication devices or input/output devices. Since the file number and channel number are of the same concept, the two must not overlap. Any constant, variable or expression can be given as the file number and channel number, as long as they are integers in the range of 1 through 16.

(2) Structure of File descriptor

File descriptors consists of character strings of the following structure:

"[<device name>:] [(<option>)] [<file name>]"

File descriptors must always be placed between two double quotes ("). In the [FORMAT] section of Chapter 2, in order to call readers' attention, the double quotes are written at the outside like ⌜"<file descriptor>"⌟ . However, the double quotes are principally parts of file descriptors and they must always be written when designating file descriptors.

Also note that when a file descriptor is omitted in LOAD or other commands, DISK BASIC assumes that "0: ␣␣␣" has been designated.

In L-3 BASIC, <device name> may be omitted when CAS0: is designated, but in DISK BASIC, if <device name is omitted, ⌜0:⌟ (Drive 0 of minifloppy disk) is assumed. Therefore, device name of other than Disk Drive 0 cannot be omitted (except LOAD?, SKIPF).

<device name>must always be followed by a colon (:) when it is designated. This is also written separately in the FORMAT section of Chapter 2, but it is better to understand and remember that <device name> does include a colon at the end.

(3) Device name

In DISK BASIC, devices as shown in the following table are defined as <device name>.

To use devices other than the keyboard, screen, printer 0, RS-232C port 0 and cassette, an interface card for each device is needed. MP-1800 (MP-1801 for drives 2 and 3) is needed for the MP-3540 minifloppy disk.

No.	Device		Device name	Input	Output	Remarks
1	Keyboard		KYBD:	o	x	
2	Screen		SCRN:	x	o	
3	Printer	0	LPT0:	x	o	
4		1	LPT1:	x	o	
5		2	LPT2:	x	o	
6	RS-232C port	0	COM0:	o	o	
7		1	COM1:	o	o	
8		2	COM2:	o	o	
9		3	COM3:	o	o	
10		4	COM4:	o	o	
11	Cassette		CAS0:	o	o	Default in L-3 BASIC
12	Floppy disk	0	0:	o	o	Default in DISK BASIC
13		1	1:	o	o	
14		2	2:	o	o	
15		3	3:	o	o	

(4) Option

Alphanumerics of up to 6 characters placed between parentheses can be designated as an option. For example, in the communication (RS-232C), mode is designated with the option.

In the DISK BASIC, the option is not provided for other than those related to control of the RS-232C line.

(5) File name

Though up to 8 characters and special symbols are permitted for the <file name> , colon (:), and characters whose code are 0 or 255, must not be used in the <file name>. The<file name> is not used in instructions of other than disk and cassette, and on all instructions to disks, excluding the FILES Instruction, the <file name> must not be omitted. When it is omitted, designation of "␣ ␣ ␣ ␣ ␣ ␣ ␣" is assumed. The same applies when " " (null string) is designated.

2.1.14 Graphic Mode

Various display resolutions can be selected for graphics of MB-6890 by combining number of horizontal display characters in character mode and resolution mode. Graphic modes that can be used with DISK BASIC are shown in the list below.

Resolution mode	Normal mode		High resolution mode	
Number of horizontal display characters	40 digits	80 digits	40 digits	80 digits
Graphic mode (width x height)	80 x 100 dots	160 x 100	320 x 200	640 x 200
Color resolution (width x height)	40 x 25 dots	80 x 25	40 x 200	80 x 200
Memory capacity for one screen	1K bytes	2K	8K	16K
Character modes concurrently usable	40 chars. x 23 lines	80 x 25	40 x 25	80 x 25

The resolution mode and number of horizontal display characters are set by the SCREEN command and WIDTH command respectively. Graphics and characters can be used concurrently on the screen. Color of the graphics can only be specified for each horizontal 8 dots in the high resolution mode and for one character area in the normal mode. Position specification in graphic instructions (PSET, PRESET, LINE, POINT and PAINT) must be given in the range of 0 through 639 in width and 0 through 199 in the height, regardless of resolution of the graphic mode.

As described in Clauses 1.5 and 1.7 of the L-3 BASIC Manual, these graphic modes cannot be shifted to the graphic mode that requires a memory area greater than the display mode determined by No.3 and No.4 DIP switches (or by the NEW ON command).

For further details, refer to Item 2.12.2 of the L-3 BASIC Manual.

2.1.15 Interlace Mode

Up to present, video display interfaces of personal computers have been a non-interlace type that displays one screen with one scanning of CRT.

DISK BASIC provides the function of Hira-gana display in addition to the character codes (JIS C6220-1976) in units of 8 as specified in JIS. In order to realize the Hira-gana display function, which is an important feature, interlace type is adopted in MB-6890. Normally, characters consist of 8 x 8 dots, but in order to express the beautiful curved lines unique to Hira-gana, a system of 8 dots in width and 16 dots in the height was adopted and one screen is obtained by two scannings, in the same way as the standard TV signals. Therefore, when the interlace mode is switched by the SCREEN command, either graphic characters or Hira-gana are selected for some character codes. For details on this point, refer to the character code tables.

When conventional displays are switched to the interlace mode, sometimes the screen flickers. For users who desire stabilized beautiful screens in the interlace mode, the C14-2170 (Color) or K12-2055P (Monochromatic) display exclusively designed for MB-6890 and having a long after-glow time of fluorescence is recommended.

2.1.16 Multipage

As described in Item 2.1.14, since the 4 type graphic modes require different memory sizes and since the maximum graphic mode is determined by the DIP switch (or NEW ON command), at display modes requiring smaller memory sizes, the excess memory can be utilized to obtain two or more display screens. The number of pages for such case is shown in the following table.

No.	DIP switch		Number of multipages			
	No. 4	No. 3	Normal mode		High resolution mode	
	Normal/High resolution	80/40	40 digits	80 digits	40 digits	80 digits
1	1	1	2 pages	1 page	x	x
2	1	0	1	x	x	x
3	0	1	16	8	2 pages	1 page
4	0	0	8	4	1	x

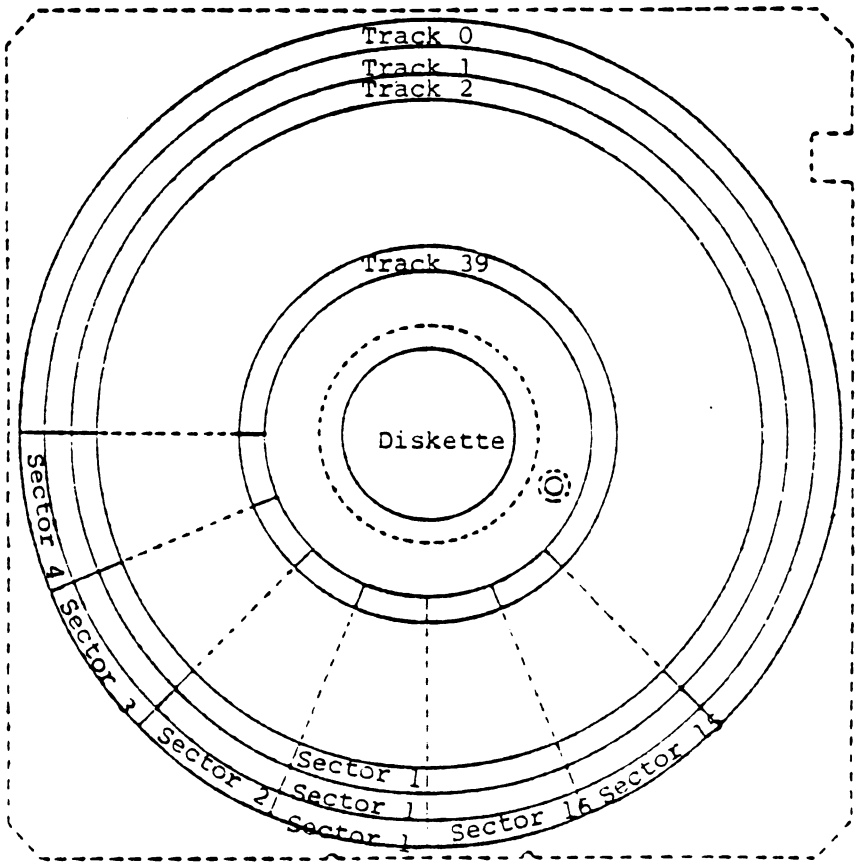
The standard state when shipped out is No. 4.

x indicates impossible.

For details of use, method of multipages, refer to the L-3 BASIC Manual.

2.1.17 Minifloppy Disk

This section describes the recording structure of the minifloppy disk. The contents are not essential for operating DISK BASIC, and readers may pass this section.



Data is recorded on the diskette concentrically, as shown in the above figure. Each concentric circle is called a track and one diskette has 40 tracks, numbered 0 through 39. One track is divided into 16 sectors numbered 1 through 16. In DISK BASIC, the area on the disk is controlled in units of group. Since the MP-3540 minifloppy disk is of one-side single density, one group consists of 4 sectors and one sector consists of 128 bytes. One sector is allocated to one record of random access files. Numbers ranging from 0 through 155 are assigned to one group consisting of 4 sectors.

The system diskette associated with this manual has the following track structure.

Track No.	Contents
0∿2	DISK BASIC code
3∿ 19	User area
20	Directory
21 ∿ 39	User area

The directory track is further divided into FAT (File Allocation Table) and a directory for each file.
FAT is a table that shows use status of each group and it resides in Sectors 1 and 2. Data written in FAT has the following meaning.

Value of data	Meaning
0 ∿ &H9B	Indicates that the data is a part of the file and the succeeding group is waited for. The value indicates the number of the succeeding group.

Value of data	Meaning
&HC0 ∿ &HC3	Indicates that the data is the last group of the file and the low order 4 bits shows the number of sectors actually used less one.
&HFE	Indicates that the area is being used by the system, and not provided for users.
&HFF	Indicates that the area is unused.

The directory of each file has the file name, the head position and attributes, and all files are processed by referring to FAT. The directory has 32 bytes for each file and the bytes are allocated as follows:

Bytes 0 through 7 : File name
Bytes 8 through 13 : Attribute
Byte 14 : The head group number where the file is stored.

Bytes 15 through 31: Spare

Refer to Chapter 3 where examples of FAT and directory are given.

Diskette format is set to this form by the FORMAT program. The directory is initialized by the DSKINI instruction. When you start using a new diskette purchased in the market or an old diskette used by other system, and when the directory or format of the existing diskette has been destroyed by an error, the diskette must be initialized using the FORMAT and DSKINI. (The system diskette has already been initialized. Never execute the FORMAT program or DSKINI Instruction to it.)

Diskette thus initialized is not recorded with the DISK BASIC codes and tracks 0 through 2 can be used by the users. However, DISK BASIC cannot be started by such diskette.

If the system diskette or a diskette that has already been used is intialized, files that have already been recorded can no longer be read. Pay particular attention not to pick up wrong diskette for initialization.

There are roughly two ways to access disks; one is by file processing, and though simple instructions are needed for the start and end of the file and delimit of individual data items, the rest are controlled by DISK BASIC.
The other way is direct access to a specific sector by users applying a program. DSKO$ and DSKI$ are for this method, and users must control all delimit of data items. Also, if the directory or file of the diskette cannot be accessed. Because of this possibility, extreme care must be paid when using the DSKO$ Instruction.

2.1.18 Different Points with LEVEL-3 BASIC

(1) Memory map

Due to extension to DISK BASIC, the user areas and memory map became different. See clauses 1.2 and 4.1

(2) Error message

For the same reason, more error messages are added. See Clause 4.3.

(3) LOAD, LOAD?, LOADM, SKIPF

When <file descriptor> is omitted in the LOAD or LOADM instruction, loading of file having the name of null string (regarded as a space) is attempted from Drive 0 of the disk. When the file name is omitted

by designating cassette as the device name, the file first found is loaded. Also, since default value of the <device name> in DISK BASIC is "0:" (Drive 0 of the disk), describe "CAS0:...." to designate the cassette.

The LOAD? and SKIPF instructions cannot be used on disks and use of these instructions is the same as with L-3 BASIC.

(4) Added instructions and functions

The following instructions have been added:

DSKINI, DSKO$, FIELD, FILES, GET, KILL, LSET, NAME, PUT and RSET

The following functions have been added:

CVD, CVI, CVS, DSKF, DSKI$, LOC, LOF, MKD$, MKI$, MKS$.

2.2 INSTRUCTIONS

2.2.1 AUTO

PURPOSE — Automatic creation of line numbers

2.2.2 BEEP

PURPOSE — Generation of sound by a built in speaker

2.2.3 CLEAR

PURPOSE — Clearance of variables and setting of character and user areas

2.2.4 CLOSE (Closing of file)

PURPOSE — Closing of files (Releases I/O buffers and channels from files.)

FORMAT — CLOSE [[#] <file number> [.[#] <file number>].....]

EXPLANATION — Instruction for final processing of the file having the <file number.> Closes that file that uses the I/O buffer (channel) of the designated number and releases the buffer for the next use. In the case of an output file (excluding random access files of the disk), this instruction outputs data remaining in the buffer.

Since there is a connection between the file and file number,both can be used for a new open instruction.

When the file number is omitted, all files of open state become the objective of the instruction.

All files are closed by executing the END statement as well.

2.2.5 CLS

PURPOSE Erase display screen

2.2.6 COLOR

PURPOSE Color designation of the screen

2.2.7 COM(n)ON/OFF/STOP

PURPOSE Permission/inhibition/stop of interrupt from the communication line

2.2.8 CONSOLE

PURPOSE Designation of scroll window

2.2.9 CONT

PURPOSE Restart of temporarily stopped program

2.2.10 DATA

PURPOSE Storing of constants for the READ statement

2.2.11 DEF FN

PURPOSE Definition of functions

2.2.12 DEF INT/SNG/DBL/STR

PURPOSE Form declaration of the designated variable

2.2.13 DEF USR

PURPOSE — Setting of the start address of the machine language user function

2.2.14 DELETE

PURPOSE — Deletion of the designated line

2.2.15 DIM

PURPOSE — Declaration of the size of array variables

2.2.16 DSKINI (Initialization of the directory track)

PURPOSE — Initialization of the directory tracks

Format — DSKINI <drive number>

EXPLANATION — DSKINI stands for DiSK INItialize, and the instruction initializes the directory tracks of the diskette.

New diskettes where nothing has been written, diskettes that have been initialized for other systems and diskettes where the directory track contents have been destroyed cannot be used in this system without executing the FORMAT program first and this DSKINI instruction to them. (See item 1.3.1 FORMAT.)

The message of ⌜Are you sure (Y or N)?⌟ is displayed when this instruction is executed. Check the diskette and drive, and input Y. The RETURN key need not be pressed.

2.2.17 DSKO$ (Direct write to disk)

PURPOSE Direct write to the disk

FORMAT DSKO$ <drive number>, <track number>, <sector number>, <character expression>

EXPLANATION DSKO$ stands for DISK Output$, and this instruction writes the specified character strings in the designated sector of the designated drive, unrelatedly to the file operation. The character expression must be for 128 or more bytes. If it is less than 128 bytes, it causes an illegal Function Call error. Any bytes exceeded 128 bytes are ignored.

No instruction like OPEN, CLOSE and others is needed.

2.2.18 EDIT

PURPOSE Calling of the designated line to the screen.

2.1.19 END

PURPOSE Declaration of program termination

2.2.20 ERROR

PURPOSE Simulation of errors

2.2.21 EXEC

PURPOSE Execution of machine language programs.

2.2.22 FIELD (Definition of record field)

PURPOSE Definition of record fields

FORMAT FIELD [#] <file number> , <field length> AS <character variable name> [, <field length> AS <character variable name> .....]

EXPLANATION This instruction defines the data structure of the designated random access file record.

One record of random access files is in a fixed length, and data in the record is also in a fixed length and excess part is trucated. However, definition of length of each data field is left with users, and data length and corresponding variable name are defined by this instruction.

Since one record must be within 128 bytes, if total of field length defined in the FIELD statement exceeds 128 bytes, it causes a Field Overflow error.

Structure of one whole record must be defined in one FIELD statement and the statement cannot be divided into two or more parts. However, two or more field structures can be defined to one record, and all of the designated field structures are concurrently valid.

To enter data into the record, the LSET or RSET statement must be used. If a PUT instruction is executed prior to execution of the LSET or RSET statement, 0 (null string) is loaded on all records.

If contents of the character variables used in the FIELD statement altered or assigned by instructions of other than LSET or RSET statement, the corresponding FIELD statement becomes invalid, and the file buffer contents do not change.

2.2.23 FILES (Display of file catalog)

PURPOSE Display of file catalogs

FORMAT FILES ["<device name>]

EXPLANATION This instruction displays the file name and its attribute on the designated device (disk or cassette) as shown in the following:

(1) In the case of disk

<file name><kind><type><file type>
<size>

(2) In the case of cassette

file name kind type

Kind
- 0: BASIC program
- 1: Data file
- 2: Machine language program

Type
- A: Character (ASCII)
- B: Internal expression (binary)

File type
- R: Random access file
- S: Sequential access file

Size: The file size is expressed in the number of groups (1 group consists of 4 sectors)

To be specific, the kind is how the data is recorded (in other words how it can be loaded). That is, kind of data recorded by the SAVE instruction is ⌜2⌟ regardless of the contents. Even if the data is in a machine language, when it is recorded by PUT or PRINT#, it falls in the ⌜1⌟ category of the kind.

When the " <device name> " is omitted, the disk of Drive 0 is selected.

2.2.24 FOR~NEXT

PURPOSE Repeating of designated count

2.2.25 GET (Read from random file)

PURPOSE Read of record from random files

FORMAT GET[#] <file number> [, <record number>]

EXPLANATION This instruction reads record of the designated number from random access files. The record contents can be referred to with the character variable defined in the FIELD statement.

When the <record number> is omitted, the LOC function is used as the record number. Initial value of the LOC function is 1, and when the record number is omitted in the first GET statement, it results in the same as designating 1 for the record number.

Note that prior to executing the GET statement, the OPEN and FIELD statements must be executed to the file designated by the <file number> .

2.2.26 GOSUB

PURPOSE Branch to subroutine

2.2.27 GOTO

PURPOSE Unconditional branch to the specified line

2.2.28 IF~ THEN/GOTO~ ELSE

PURPOSE Control of program flow depending on condition judgement

2.2.29 INPUT

PURPOSE Input from the keyboard

2.2.30 INPUT# (Input from sequential file)

PURPOSE Input from sequential files

FORMAT INPUT #<file number> , <variable name> [, <variable name> .....]

EXPLANATION This instruction reads data from the sequential file designated by the <file number> and assigns the contents to the variable. In order to execute the INPUT# statement, the OPEN statement must be executed in advance in the input mode ("1").

The sequential file mentioned here means all input devices excluding random files.

CR (Character code 13. &HOD), comma (,) and colon (:) are used for delimit of data items and these characters cannot be read as data.

2.2.31 INPUT WAIT

PURPOSE Input from the keyboard with time limitation

2.2.32 KEY

PURPOSE Definition of programmable function key

2.2.33 KEY LIST

PURPOSE Output of definition contents of the programmable function key

2.2.34 KEY ON/OFF/STOP

PURPOSE Permission/Inhibition/Stop of interrupt of the programmable fucntion key

2.2.35 KILL (Deletion of file)

PURPOSE Deletion of files

Format KILL " ‹file descriptor› "

EXPLANATION This instruction deletes the designated file from the disk. One KILL instruction can delete only one file, and the file to be deleted must not be in open state.

The KILL Instruction is effective too all kind and type disk files (BASIC program files, machine language program files, sequential access files and random access files).

2.2.36 LET

PURPOSE Assignment of an expression value to a variable.

2.2.37 LINE

PURPOSE Drawing of a line between designated two points.

2.2.38 LINE INPUT

PURPOSE One line input from the keyboard

2.2.39 LINE INPUT# (One line input from sequential file)

PURPOSE One line input from sequential files.

FORMAT LINE INPUT #<file number> , <character variable name>

EXPLANATION

This instruction reads into the character variable from the designated sequential file (all input devices excluding random files). Function of this instruction is same as that of the INPUT# instruction, except the following:

- One instruction can read into only one variable.
- The data must be in character type.

- All characters up to [CR] are effective as data. [CR] is not received as a data item. The rule to limit the number of characters to one variable up to 255 bytes is effective.
- End of data cannot be determined by the EOF function.

2.2.40 LIST (Output of program file)

[PURPOSE] Output of program list to designated output file

[FORMAT] LIST [" <file descriptor> "] [,[< start line number>][| - | or | , | [< last line number>]]]

(Abbreviation form is L.)

[EXPLANATION] This instruction outputs a part or whole of the program residing in the memory at present. When the <file descriptor> is omitted, the file contents are output to the screen. Omission of the start line number results in the same thing as designating the program head, and omission of the <last line number> results in the same things as designating the program end. Period (.) may be used instead of each line number. If (.) is used, the line to be displayed is the last line to be executed by the list statement, immediately prior to the error occurring, or a STOP or END statement encountered.

When LPT1, LPT2, or COM1 through 4 is designated as the device name of the file descriptor, if there is no pertinent I/O card in the extended interface section, it causes an error.
Designation of disk or cassette for the device name results in the same thing as SAVE of ASCII type.

2.2.41 LOAD (Read of program)

PURPOSE — Read of programs from disk or cassette

FORMAT — LOAD ["〈file descriptor〉"] [,R]
(Abbreviation form is LO.)

EXPLANATION — This instruction loads the program file designated by the cassette or disk into the memory. When the 〈file descriptor〉 is omitted, Drive 0 of the disk is assumed for the device and the file name becomes null string (same as the space).

When the LOAD instruction is executed, first, the program in the memory is erased and the variables lose their values.

During LOAD execution from a cassette, if CTRL + D are keyed in before the designated program is found, the execution is stopped and the program on the memory is retained.

When the ⌜, R⌟ of the second parameter is designated, the program is executed as

soon as it is loaded. Also, files that are in open state at the time are not closed.

2.2.42 LOAD?

PURPOSE — Check of program on the cassette

2.2.43 LOADM (Read of machine language program)

PURPOSE — Read in of machine language program or data from disk or cassette

FORMAT — LOADM ["<file descriptor>"] [,<offset>] [,R]

EXPLANATION — This instruction reads machine language program or others (recorded by SAVEM instruction) recorded in the disk or cassette onto the memory.

When the <offset> is designated, the program contents are loaded onto the address which is the addition of the address at the recording time and the offset value.

When the [,R] is designated, the machine language program is executed as soon as it is loaded. The execution starts at the address designated during SAVEM execution.

Pay attention not to load the program contents to the area where the DISK BASIC codes are entered when using this instruction.

2.2.44 LOCATE

PURPOSE — Designation of the cursor position and function

2.2.45 LSET or RSET (Storing of data to random file)

PURPOSE — Data storing in random file buffer in left or right justified form

FORMAT — LSET<character variable name> = <character expression>
RSET<character variable name> = <character expression>

EXPLANATION — LSET, RSET stands for Left SET, Right SET, and this instruction stores data in the area corresponding to the character variable that was defined in the FIELD statement of the random access file record. The data to be stored must be character type data and any numeric data must be converted to character type data by either one of MKI$, MKS$ or MKD$.

When the number of characters in the <character expression> is longer than the length allocated by the FIELD statement, the data is entered in the field in left-justification for LSET and right-justiciation for RSET. If the character expression is longer than the field, the right side part is lost both in LSET and RSET.

The character variable name must have been defined by the FIELD statement. If a variable that has not been defined is used, it causes a String Not Fielded error.

2.2.46 MERGE (Combining of programs)

PURPOSE Merge of current program with a program from the disk or cassette.

FORMAT MERGE" <file descriptor>" [,R]

EXPLANATION This instruction merges a program currently on the memory with a program on the disk or cassette and stores the result on the memory.

The program on the disk or cassette must be in character type (ASCII format- refer to SAVE Instruction and FILES instruction).

Line number of the program on the disk or cassette may be younger than that of the program on the memory, or the two may be crossing, but if the two programs has the same line number, data of the line on the memory is deleted and the line on the disk in given with the priority.

If there is a MERGE Instruction in the program, the system returns to the command level after executing the MERGE instruction.

When the ⌜,R⌟ is designated, the program is executed from the head after execution of the MERGE instruction.

2.2.47 MID$ (Partial replacement of character variable)

PURPOSE — Partial replacement of character variable contents.

FORMAT — MID$ (<character variable name> , <argument> 1 [, <argument 2>]) =<character expression>

EXPLANATION — This instruction replaces the number of the characters as specified by the <argument 2> starting from the position as specified by the <argument 1> in the designated character variable with the contents of the <character expression>. If the <character expression> is longer than the <argument 2>, excess at the right side is ignored. For the value of the <argument 2> , the smallest value among the three of <argument 2> , length of the <character variable name> after the position of the <argument 1>, and the <character expression> length.

The <argument 2> must be in the range of 0 through 255. The <argument 1> must be in the range of 1 through 255 and also not longer than the designated character variable length.

2.2.48 MON

PURPOSE — Activation of the machine language monitor.

2.2.49 MOTOR

PURPOSE — Motor control of cassette

2.2.50 NAME (Name alteration)

PURPOSE — Alteration of file name

FORMAT — NAME" <current file descriptor> " AS " <new file descriptor> "

EXPLANATION — This instruction alters the name of the file on the disk. It cannot be used on cassettes. If a file having the same name as the file name designated in the <new file descriptor> exists on the same diskette, it causes a File Already Exists error.

2.2.51 NEW

PURPOSE — Deletion of a program and clearing of all variables.

2.2.52 NEW ON

PURPOSE — Restart by mode switching.

2.2.53 ON~ GOTO/GOSUB

PURPOSE — Branch (to a subroutine) by the expression value

2.2.54 ON COM(n) GOSUB

PURPOSE — Designation of start line of interrupt routine originated by the communication line

2.2.55 ON ERROR GOTO

PURPOSE — Designation of start line of error handling routine

2.2.56 ON KEY (n) GOSUB

PURPOSE — Designation of start line of interrupt routine caused by key-in

2.2.57 ON PEN GOSUB

PURPOSE — Designation of start line of interrupt routine caused by light pen operation

2.2.58 OPEN (Opening of file)

PURPOSE — Opening of file (allocation of input/output buffer and channel to a file)

FORMAT — OPEN " <mode> ", [#] <file number> , "<file descriptor> "

EXPLANATION — This instruction initializes I/O operation. It defines the number of the file designated in the <file descriptor> , preserves the buffer, and sets the file pointer.

There are the following three types for the <mode>:

"I" - input mode
"O" - output mode
"R" - random access mode

The file pointer is set at the file head in the input mode. In the output mode and random access files, if the designated file already exists, the file pointer is set at the file. If it does not exist, an open area is searched and the file pointer is set at the area head. In the input mode, if the designated file does

not exist, it causes an error.

The screen and printer can be opened only by the output mode, and the keyboard by the input mode only.

Instructions such as FIELD, GET, INPUT , LINE INPUT#, LSET, RSET, PRINT# and PUT and functions such as INPUT$, LOC and LOF accompanying EOF and a file number cannot be used before the pertinent file is opened. Other instructions and functions (DSKO$, DSKI$) can be executed without the OPEN statement execution.

Up to 16 files can be concurrently opened and the number can be changed by the CONFIG utility program. Up to 40 files can be recorded in one diskette, and the number is limited by the minifloppy disk capacity as well.

Files in open state must be closed before the diskette is taken out of the drive. Other-wise, the contents cannot be guaranteed.

When a file is opened in the output or random access mode, if another file having the same name exists, data is output by destroying the existing file contents, regardless if the file is a data file or program file.

2.2.59 PAINT

PURPOSE Coloring of designated area

2.2.60 PEN

PURPOSE — Definition of light pen area

2.2.61 PEN ON/OFF/STOP

PURPOSE — Permission/inhibition/stop of light pen interrupt

2.2.62 POKE

PURPOSE — Write of data to the designated address in the memory

2.2.63 PRESET

PURPOSE — Erase dot at the designated position

2.2.64 PRINT

PURPOSE — Output to the display

2.2.65 PRINT USING

PURPOSE — Format control output to the display

2.2.66 PRINT# (Format control output to sequential file)

PURPOSE — Output (format control) to sequential file

FORMAT — PRINT#<file number> [,<expression>][; or , <expression> .....]

PRINT#<file number>, USING <format expression>;<expression>[; expression or , ...]

EXPLANATION

This instruction outputs data to the designated sequential file. The sequential file mentioned here includes the printer, RS-232C port and screen, in addition to sequential files of both the disk and cassette files.

Description of USING provides format control. For details of the format expression contents, refer to the LEVEL-3 BASIC Manual.

When outputting to the printer, sometimes there is a timing difference between execution of PRINT and actual print out. To make sure of output at the time of execution, execute ⌜PRINT #n⌟ in succession.

Prior to executing the PRINT statement, the file must be opened in the output mode ("O"). Also, sometimes data is not actually output if the file is not closed at the end of the processing.

2.2.67 PSET

PURPOSE

Setting of a dot at the designated position

2.2.68 PUT (Output to random file)

PURPOSE

Writing of record to a random file

FORMAT

PUT [#] < file number > [, record number]

EXPLANATION

THis instruction writes a random file record into the disk, marking the record

with the designated <record number>.

If the <record number> is omitted, the next record number of the record accessed by the PUT or GET statement that was executed immediately before, that is contents of the LOC function, is applied.

Prior to executing the PUT statement, the file must be opened, the record configuration must have been defined by the FIELD statement, and contents of each field must have been determined by either LSET or RSET. If the OPEN and FIELD statements have not been executed, it causes an error. If either LSET or RSET does not exist but if either one of them has been executed in the past, data at the time is output. If neither has been executed, 0 (null string) is output.

If a greater number than 624 is given as the <record number>, a Bad Record Number error occurs.

2.2.69 RANDOMIZE

PURPOSE Alteration of random number system

2.2.70 READ

PURPOSE Read from data statements

2.2.71 REM

PURPOSE Annotation

2.2.72 RENUM

PURPOSE Renumbering of lines

2.2.73 RESTORE

PURPOSE Designation of read start line from data statement

2.2.74 RESUME

PURPOSE Return from error handling program

2.2.75 RETURN

PURPOSE Return from subroutine

2.2.76 RSET (Storing of data to random file)

See 2.2.45 LSET. RSET.

2.2.77 RUN ((Load) and execution of program

PURPOSE Start of (load) and exe ution of program

FORMAT RUN[<line number>] (Abbreviation form is R.)

RUN"<file descriptor>"[,R]

EXPLANATION This instruction executes the BASIC program from the youngest line number if the parameter is not described. If described, the execution start from that line.

If the file descriptor is designated, the file of the designated disk or cassette is loaded and executed immediately.

Prior to starting execution of the program, previous programs in the memory are erased, numeric variable contents are cleared to [0] , and character variables are cleared to null string. Also, all open files are closed.

However, when the R option is designated, open files remain in the previous state and are not closed.

When the RUN" file descriptor " is executed to a cassette, programs on the memory are kept until the designated file is found, and the program search can be stopped by pressing CTRL and D keys.

2.2.78 SAVE (Recording of program)

PURPOSE Recording of program to disk or cassette

FORMAT SAVE["<file descriptor>"] [,A]
(Abbreviation form is SA.)

EXPLANATION

THis instruction records the program in the memory giving a file name to the device designated in the file descriptor .

When the [, A] parameter is specified, the program is saved in character form (ASCII format), and when nothing is specified, the program is recorded in internal expression form (binary format).

If there is another file having the same name on the same device, the previous file contents are destroyed and save, regardless of if it is a program file or data file.

2.2.79 SAVEM (Recording of machine language program)

PURPOSE Recording of machine language program or data to disk or cassette

FORMAT SAVEM"<file descriptor>", <head address> <last address>,<starting address>

EXPLANATION This instructor saves machine language programs or others on the memory to the device designated in the <file descriptor> giving a file name. Designate the head and last addresses of the memory to be saved, and starting address of executing the LOADM" <file descriptor> ",R.

2.2.80 SCREEN

PURPOSE Designation of screen mode and page.

2.2.81 SKIPF

PURPOSE Skip reading until the designated file on the cassette is found.

2.2.82 STOP

PURPOSE Holding of program execution

2.2.83 SWAP

PURPOSE Swapping of two variable values

2.2.84 TERM

PURPOSE Setting of the system to terminal mode.

2.2.85 TRON or TROFF

PURPOSE Turning on/off of trace mode

2.2.86 WIDTH

PURPOSE Setting of the number of display characters in one line.

2.3 FUNCTIONS AND SYSTEM VARIABLES

2.3.1 ABS

PURPOSE To give an absolute value.

2.3.2 ASC

PURPOSE To give a character code corresponding to a character.

2.3.3 ATN

PURPOSE To give an arc tangent.

2.3.4 CDBL

PURPOSE Double precision conversion

2.3.5 CHR$

PURPOSE To give a character corresponding to a character code.

2.3.6 CINT

PURPOSE Integer conversion

2.3.7 COS

PURPOSE To give a cosine.

2.3.8 CSNG

PURPOSE Single precision conversion

2.3.9 CSRLIN

PURPOSE — To give a vertical position of the cursor.

2.3.10 CVI, CVS, CVD (Conversion to numeric data)

PURPOSE — Restores character type numeric data in the random buffer to the numeric type data.

FORMAT

CVI(<character string of 2 bytes>)
CVS(<character string of 4 bytes>)
CVD(<character string of 8 bytes>)

EXPLANATION — CVI, CVS, CVD stands for ConVert to Integer (or Single or Double) number. Data stored in the random file must be character type variable, and numerics are converted to character type by MKI$, MKS$ or MKD$ function. Data thus converted to character type is converted back to the original numerics by CVI, CVS or CVD. CVI applies to integers, CVS to single precision real numbers and CVD to double precision real numbers. For details, refer to the EXPLANATIONs of MKI$, MKS$ and MKD$.

2.3.11 DATE

PURPOSE — To give the number of days of the internal clock. (assignment not permitted)

2.3.12 DATE$

PURPOSE — Internal clock data (assignment permitted)

2.3.13 DSKF (Unused area of diskette)

PURPOSE — To give a free area of the diskette.

FORMAT — DSKF (<drive number>)

EXPLANATION — DSKF stands for DISK Free, and the function returns unused area of the diskette inserted in the drive that is designated in the <drive number> in units of the group. In the one-side single density minifloppy, one group consists of 4 sectors (=512 bytes).

2.3.14 DSKI$ (Contents of disk sector)

PURPOSE — To give contents of the designated sector in character strings.

FORMAT — DSKI$(<drive number>,<track number>, <sector number>)

EXPLANATION — DSKI$ stands for DiSK Input$, and the function gives contents of the designated drive, track and sector in character strings. Since this function is not related to file operation, no procedure like OPEN is needed. In the case of one-side single density minifloppy, one sector consists of 128 bytes, so that the contents can be received in one character variable.

2.3.15 EOF (End of data)

PURPOSE — Determination of data end of a disk or cassette file

From now on, explanation is limited to data exchange with a disk.

A unit called record is used to read/write data with data files.

Let us take up a file of addresses. The name, address, telephone number, etc. are put together as data for each person. This group of data for each person is called a record. In the case of stock management programs, article name, stock quantity, refill level, unit price, etc. are put together as data for each article, that is, record.

Data files are divided into two types depending on how they are wirtten and read; sequential access file and random access file.

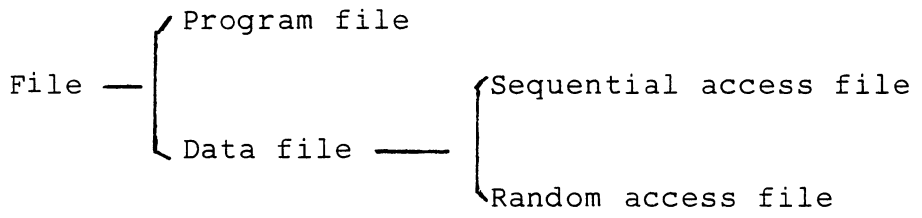


Fig. 2. File Structure

In the system of random access files, information to indicate location of specific records in the disk is provided separately, and data can be immediately read by referring to the information, no matter where the record exists.

On the other hand, sequential access files can only be read in the sequence of write. Therefore, even if data located in the middle part of a file is needed, the file must be read from the start until finding the objective data. Also, in sequential access, the concept of record is not clear and delimit between a record and another is obscure. However, this gives opportunities of setting file

forms freely, and since the programs can be simpler, the sequential access file system does have an advantage on certain uses.

Regardless of the type, sequential access or random access, all data files must be opened before they are used. As explained before, the buffer is set by the OPEN instruction, and so is a file pointer. Setting of a file pointer means search of the objective file when the OPEN instruction is executed in the INPUT mode (that is, to read the file) and to memorize the position. In the OUTPUT mode (that is, to write into the file), a blank area is searched in the disk and the position is memorized.

When use of the file terminates, the CLOSE instruction must be executed as a rule. The buffer is separated from the file by the CLOSE instruction, and in the OUTPUT Mode, the last data is written into the disk (except random files).

3.2 PROGRAM FILE OPERATION

This section describes input/output instructions of program files, but prior to it initialization of disks is briefly explained.

Diskettes available in the market cannot be used for read/write with the DISK BASIC, so the FORMAT program and DSKINI instruction must be executed onto them. For detailed procedures, refer to 1.3.1 Format (Diskette initialization) and 2.2.16 DSKINI.

When the diskette contents are destroyed by mistake, sometimes execution of the DSKINI instruction alone may restore the diskette, but the recorded data can never be restored to the normal state. The data can be read by the DSKI$ function but unless the contents of the directory track (see 2.1.17 Minifloppy Disk) are compensated, it cannot be read by other instructions and functions.

See Program 1 below. This is a program to register a name, address and telephone number as an address catalog to a file. Details are not explained at this moment as 3.3.1 gives further explanation of it, but since it is used as example to explain the program file operation, key in the program first.

Program 1

```
1000 OPEN "O",#1,"0:ADDRESS"
1010 INPUT "NAME";NAME$
1020 IF NAME="END" GOTO 1070
1030 INPUT "ADDRESS" ADDRESS$
1040 INPUT"TELEPHONE"; TEL$
1050 PRINT#1,NAME$;",";ADDRESS$;",";TEL$
1060 PRINT:GOTO 1010
1070 CLOSE#1
1080 END
```

Then, preserve this program in the disk. To do so, key in as follows:

```
SAVE "0:ADDRESS"
```

ADDRESS is the file name and 0: is the device name. The unit of "0:ADDRESS" is the file descriptor. As explained before, keying in of SAVE "ADDRESS" alone does not clarify whether it is to be saved in a cassette or in a disk. The device is designated in the file descriptor to clarify this. The 0: refers to the disk drive of No. 0. The device name can be omitted only when No. 0 disk drive is designated.

Up to 8 characters may be used for the file name, and practically all characters, except colon (:) can be used.

When the key-in operation as shown in the preceding page is performed, a sharp sound of the drive head moving is heard as the program is recorded on the disk. During the time the head is touching the diskette, the red LED lamp at the left bottom of the Drive 0 cover lights up. Since the access time to the disk is very short, to human eyes, the LED lamp looks as if it blinked a few times.

If the drive motor is stopped (the motor lamp is not on) during the time of executing the SAVE Instruction, the motor is automatically set ON to move the head. The lamp which was not on lights up also.

Since the disk head is writing the program into the diskette when the red LED lamp of the drive is lit, never open the cover, nor take out the diskette.

This completes writing of Program 1 into the diskette.
Once the program is saved in the disk, there is no need of keying in the same program again.
Then, let us call out this program. First, erase any program

that remains in the unit using the NEW instruction. Naturally, even if a program remains in the memory of the unit, it is automatically erased by the LOAD instruction. We try this erase operation just to check that Program 1 is called out from the disk. Use the LIST instruction to make sure that no program remains in the unit and execute the LOAD instruction to call out the program:

```
LOAD "0:ADDRESS"
```

The "0:ADDRESS" is a file descriptor as explained before. Execute the LIST Instruction and check that the program has been entered.

Next is operation of this program, by keying in:

```
RUN
```

When NAME is displayed, key in any name. Hira-gana, Kata-kana or alphabetics can be used. Key in an address and telephone number. ⌜NAME⌟ is displayed after skipping a line. Repeat the same key-in procedures for the number of persons required. When everyone's information has been keyed in, key in ⌜END⌟ to the last ⌜NAME⌟ display. When thelast data is written in the disk, execution of the program terminates.

Sample run 1

```
RUN
NAME      ? JOHN SMITH
ADDRESS   ? 15 WEST STREET, MELBOURNE
TELEPHONE ? (03) 062-4952
NAME      ? END

Ready
```

There are the following two ways to execute programs contained in the disk, in addition to keying in RUN after loading:

(1) LOAD "0:ADDRESS", R

(2) RUN "0:ADDRESS"

In either case, LOAD and RUN are both executed by one instruction.

By the way, a file name of ⌜ADDRESS⌟ was given to Program 1. If the name is not liked, the file name can be changed using an instruction instead of repeating LOAD and SAVE. Since there will be many more programs using the same address catalog, giving a number at the end of "0:ADDRESS" may be a good thing, which can be realized by keying in as follows:

```
NAME "0:ADDRESS" AS "0:ADDRESS 1"
```

The sequence of the parameters is first the current file descriptor and the file descriptor of the new name follows it.

The DSKF function is provided to inform remaining capacity in the disk. key in as follows:

```
PRINT DSKF (0)
```

The number displayed indicates the number of groups of empty areas of the disk. Group is the unit of disk capacity and one minifloppy disk has the capacity of about 156 groups.

Use LOADM and SAVEM for input/output of machine language programs.

Use the FILES Instruction to check files contained in the disk.

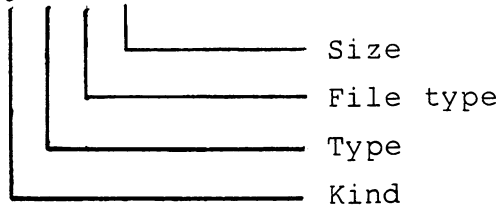
```
FILES "0:"
```

"0:" is the file descriptor. In the case of this instruction, naturally no file name is needed. When this instruc-

tion is executed, information related to the file contained in the diskette inserted in Drive 0 is displayed. File name, kind, type, file type and size are displayed on all files contained in the diskette.

Sample run 2

```
    FILES "0:"

FORMAT    0 B S 1
SKIP2     2 B S 2
CONFIG    0 B S 1
DISKCOPY  0 B S 1
ADDRESS   1 A S 1
ADDRESS 1 0 B S 1
```




```
Ready
```

The kind is difference of the file, that is, BASIC program, machine language program or data file.

The type indicates if the file is in ASCII code type or internal expression type. The ASCII code is an expression type of characters determined as standard, and correspondence between the codes and characters is widely known. Files made in the internal expression type may not be loaded with some improved BASIC versions.

The file type is indication whether the file is a sequential access file or random access file.

The size indicates the file size. The unit used is a group.

3.3 SEQUENTIAL ACCESS FILE

3.3.1 Output of Sequential Access File

Program 1 used in 3.2 creates sequential files. This section describes the output instruction of sequential files using Program 1 as an example.

The file is being opened at Line No. 1000. ⌜"O"⌟ of the first parameter is for mode, and O designates the output mode, that is a mode to write into the disk. ⌜Comma⌟ , written next is the delimit of the parameters. ⌜#1⌟ of the second parameter is for the file number and this designates that the file number of ⌜#1⌟ must be used instead of the file name of ⌜ADDRESS⌟ in succeeding I/O instruction. Therefore, on output instructions to be used from now on, designation of the file name is not needed and the file number is the only designation needed. #1 functions to connect the file and buffer at the same time. The buffer numbered as 1 cannot be used for purposes other than outputting the file named as ⌜ADDRESS⌟ Until the file is closed later. The last parameter is the file descriptor.

⌜NAME⌟ is displayed by the INPUT instruction of Line No. 1010, and input from the keyboard is waited for. Character string input is stored in the variable named as ⌜NAME$⌟ .

End of the data is determined at the next 1020. If data really ends (if ⌜END⌟ is keyed in), execution goes to Line 1070.

On Line 1030, the address is input, and on Line 1040, the telephone number. These inputs are stored in variables of ADDRESS$ and TEL$ respectively.

On Line 1050, the input name, address and telephone are written into the disk. The first parameter of ⌜#1⌟ is the file number and all others are variable names. Since the

OPEN instruction has already defined that File No.1 is to be used for output of the file, named as ⌜ADDRESS⌟ , in Disk 0, all designation needed here is just the file number. ⌜","⌟ placed between data items is a delimit sign, and without it, total of NAME$, ADDRESS$ and TEL$ is regarded as one data item. ⌜;⌟ is also a delimit sign of data items. Difference is ⌜","⌟ is used on the disk and ⌜;⌟ is a delimit sign used on programs. Since the delimit sign on the disk is automatically added at the end of the PRINT statement, Line 1050 can be divided into the following three lines:

```
1050  PRINT #1, NAME$
1051  PRINT #1, ADDRESS$
1052  PRINT # 1, TEL$
```

In this case no ⌜","⌟ should be written.

On Line 1060, a space line is given before receiving input for the next person (PRINT Instruction), and execution returns to Line 1010 to receive input again.

From Line 1070 on, execution is for termination processing, and when data end is determined on Line 1020, the execution goes to Line 1070.

Line 1070 closes the file. When terminating file processing, the CLOSE instruction must always be executed. By executing the instruction, the last data or a mark to indicate the file end is written in the disk and the buffer and file connected by the OPEN instruction are separated. Therefore, if the CLOSE Instruction is forgotten, the file is not created correctly, and it cannot be read later.

3.3.2 Input of Sequential Access File

Program 2 is a program to read the file prepared by Program 1 and display it on the screen. This section describes input of sequential access files using the Program 2 example.

Program 2

```
2000 OPEN "I",#1,"0:ADDRESS":I=1
2010 IF EOF (1) GOTO 2070
2020 INPUT#1,NAME $, ADDRESS$, TEL$
2030 PRINT "NAME  :";NAME$
2040 PRINT "ADDRESS:";ADDRESS$
2050 PRINT "TELEPHONE:";TEL$
2060 PRINT:I=I+1:IF I<7 GOTO 2010
2063 I=1:PRINT "PRESS ANY KEY ";
2065 A$=INPUT$(1):PRINT:PRINT:GOTO 2010
2070 CLOSE#1:PRINT"END"
2080 END
```

First, the file is opened by Line 2000. The parameters are the same as in the case of output, and they are in the order of mode for input or output, file number and file descriptor.

Data end is determined on Line 2010. In the preceding section for output, explanation was given to key in ⌜END⌟ to tell data end when all data has been input. It is possible to record a data item having the name of ⌜END⌟ On the file to indicate data end, but in the case of input into data file, a convenient function called ⌜EOF⌟ is provided. Therefore, let us use it. Contents of the parentheses that follow ⌜EOF⌟ indicate the file number. Therefore, what Line 2010 means is to jump the execution to Line 2070 when data of No. 1 file terminates.

Omission of this instruction causes an ⌜Input Past End⌟ error.

Beware that the file cannot be closed when this error occurs.

This instruction is subjected to a restriction related to the position on the program, that is, the instruction must

be within a loop existing between data read and processing. If it is outside of the loop, it causes an ⌜Input Past End⌟ error even if it is written.

On Line 2020, name, address and telephone number are read from the sequential access file into variables of NAME$, ADDRESS$ and TEL$ respectively. The parameters are written in the order of the file number and variable names to be read into.

What has to be noted at this point is the correspondence between the data and variables to be read into. As explained in 3.1, input is received only in the sequence of output with sequential files. Accordingly, since output in Program 1 was in the order of name, address and telephone number, the same order is followed in reading for input. Therefore, even if Line 2020 is written as follows:

```
2020 INPUT#1, ,ADDRESS$, NAME$, TEL$
```

the sequence of data coming into does not change. This means the name is entered to ADDRESS$ and the address to NAME$.

Furthermore, suppose that the following is added to Line 2025:

```
2025 INPUT#1,NAME$
```

What will happen is after reading in one person's name, the next data to be read will start from an address. Accordingly, an address will enter to NAME$, a telephone number to ADDRESS$, and name of the next person to TEL$.

In other words, use of the same variable names in input as used in output does not necessarily mean they will be read into the output variables. Correspondence between data that is read and variables is determined by the sequence of output.

With this explanation, now the reasons should be clear to understand that Line 2020 can be written in 3 lines as shown below:

```
2020 INPUT#1, NAME$
2022 INPUT#1, ADDRESS$
2024 INPUT#1, TEL$
```

This applies to output also, and writing in one line or writing in two or more lines by splitting the contents result in the same thing.

Sample run 3

```
   RUN
   NAME     : JOHN SMITH
ADDRESS     : 15 WEST STREET, MELBOURNE
TELEPHONE   : (03) 062-4952

   NAME     : JOE BROWN
ADDRESS     : 1 NORTH ST. ADELAIDE
TELEPHONE   : (08) 098-7654

  END
Ready
```

Data that has been input is displayed on the screen by lines from 2030 to 2050.

One space line is placed by Line 2060 and display returns to the input statement of Line 2010.

If there are 7 digits in data of Lines 2060 through 2065, data is collected for each of the seven digits on the screen, and the next data is displayed upon key input.

When the data terminates, operation jumps to Line 2070 by the instruction on Line 2010. The file is closed by Line 2070. "#1" is for the file number.

Line 2080 declares end of the program.

The file number in the OPEN statement of Line 2000 needs not be No.1, but the same file number must be used consistently in the OPEN, INPUT#and CLOSE statements. A number in the range of 1 through 16 can be used for the files.

There are several subjects that were not explained in the section of program file input/output instruction, and here ⌜MERGE⌟ is explained. This is an instruction to combine two programs.

Since Program 2 is on the memory right now, let us try to merge a program output from a file created before. One thing that should be remembered is the limitation that programs must have been saved in the ASCII code form if they are to be merged (see EXPLANATION of the FILES instruction). Program 1 has been saved in the internal expression form and it cannot be immediately merged. Therefore, the current program must be saved in the ASCII format first. Key in as shown below:

```
SAVE "0:ADDRESS 2",A
```

The last parameter, ⌜,A⌟,| designates saving in the ASCII format. No designation of ASCII or internal expression is needed for LOAD.

Then load Program 1.

```
LOAD "0:ADDRESS 1"
```

Now, Program of ⌜ADDRESS 1⌟ is on the memory and the program of ⌜ADDRESS 2⌟ to merge has been saved in the ASCII format. MERGE can be executed by keying in as follows.

```
MERGE "0: ADDRESS 2"
```

When ⌜Ready⌟ is displayed, execute LIST. Program 1 should have been added.

The next operation is to set these two programs to subroutines, so that they can be registered or displayed at any time desired. First, change Lines 1080 and 2080 as follows to make the two routines to two subroutines.

```
1080 RETURN
2080 RETURN
```

The main routine that controls these two subroutines is in Program 3.

Program 3

CATALOG (PROGRAM C)

```
100 REM CATALOG (PROGRAM C)
110 CLS
120 PRINT TAB (30);"*** CATALOG ***"
130 LOCATE 10,3
140 PRINT "1: CREATE FILE"
150 LOCATE 10,6
160 PRINT "2 DISPLAY FILE"
170 LOCATE 10,9
180 PRINT "3 QUIT"
190 LOCATE 10,12
200 INPUT "ENTER NO.";A
210 IF A<1 OR A>3 GOTO 200
220 IF A=3 THEN END
230 ON A GOSUB 1000,2000
240 GOTO 110
```

When this program is run, ⌜ ENTER NO.? ⌟ is displayed on the screen and the system waits for keying in. If ⌜1⌟ is input, the subroutine from Line 1000 is executed and the address catalog file is registered. If ⌜2⌟ is input, file that has been registered is displayed. If ⌜3⌟ is input, the program execution completes and BASIC returns to the state of waiting for command.

3. DISK OPERATION

This chapter describes the disk operation and functions added to LEVEL-3 BASIC. However, since the purpose of this chapter is introduction for clear understanding of the contents in Chapters 1 and 2, accurate definitions of terms and detailed specifications of instructions may not be explained sometimes. For details, refer to reference books and Chapter 2.

Description in this chapter is based on use of the BASIC mode and 80-character mode. Set the DIP switch accordingly, or try to execute commands like ┌NEW ON 15┘. Depending on the DIP switch and NEW ON command state, there may be trouble like Hira-gana characters not being displayed.

Also note that operation of the [RETURN] key is omitted in this explanation. Unless otherwise specified, the [RETURN] key must be input at line end or in input.

3.1 FILE

Programs created are saved in a cassette tape in LEVEL-3 BASIC. Each saved program is called a file.
Also, contents of variables are preserved in a cassette tape. A group of data thus preserved is also called a file.

Thus, a file is a group of data that has meaning, and a file in which a program is written is called a program file. A file in which data is written is called a data file.

In the case of programs, one program is written in a file, but in the case of data, it is not easy to know what quantity of data can be contained in a file. This is because the file size is defined by users. In other words, data output to a file of the same name during the time from opening the file up to losing it makes one file.

In MB-6890, this file conception is extended, and input/output to and from the printer, keyboard, screen and RS-232C port are treated under the concept of creation and read of files. Accordingly, all input/output devices can be accessed by one instruction.

However, if one instruction covers all devices, it is not clear which of the devices is to be used. The facility called file descriptor is set to clarify this, and file names and device names are designated by the file descriptor.

By the way, a facility called buffer is used by computers in input/output operation. For example, data input from the keyboard is stored in the keyboard buffer, one character after another. When the RETURN key is pressed, the computer judges the buffer contents, and if what was input is a command, the computer executes it. If it is a program, the computer stores the lines in program areas.

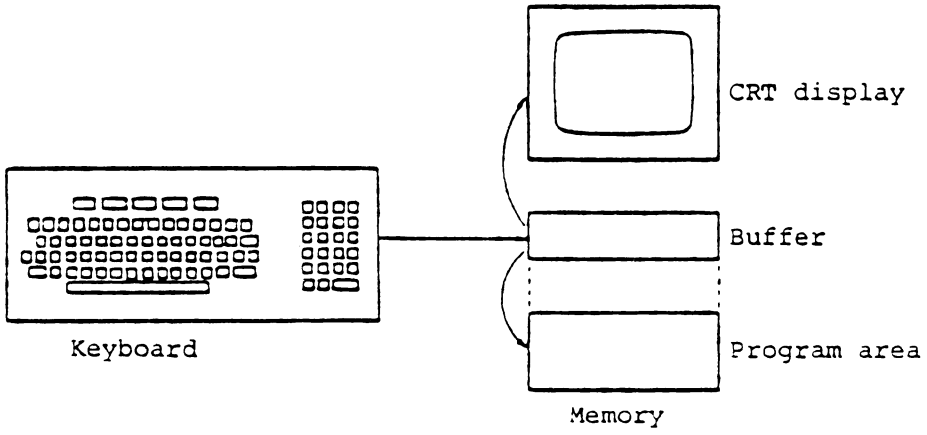


Fig. 1 Buffer

The same applies to input/output of disks and printers. I/O data is firstly stored in a buffer, and input data is then stored in a program area or variable area in units of line or data, and output data is transferred to individual devices.

Sample run 4

```
                *** CATALOG ***

    1 CREATE FILE

    2 DISPLAY FILE

    3 QUIT

    ENTER NO. ? 2
```

Save this program also. The file name given is ⌜ADDRESS 3⌟.

Describe the file name of the program that is read from the disk and merged to the current program in the file descriptor of the MERGE Instruction.

The program to be read and the current program must not have the same line number. If there is a same line number, the program to be read is give the priority. For example, if there is a line numbered 100 in the program on the memory and in the program to merge, Line 100 of the current program is erased and contents of Line 100 of the program read from the disk are recorded.

3.3.3 Modification of Sequential Access Files

This section describes addition, correction and erase of data on sequential access files. In the sequential file system, a file that is made cannot be modified. Therefore, a new file must be made after correcting the existing file. Program 4 is a program that modifies data of an address catalog file.

Program 4

```
3000 REM ***** FIND A FILE *******
3010 ON ERROR GOTO 3060
3020 NO=1:SEQ=0:PREV=0
3030 OPEN "I",#1, "0:ADDRESS"
3040 GOTO 3110
3050 REM **WHEN YOU DON'T HAVE AN "ADDRESS"**
3060 IF ERR<> 63 THEN ON ERROR GOTO 0
3070 RESUME 3080
3080 SEQ=SEQ+1:NO=NO+1:IF NO>2 THEN NO=1
3085 IF SEQ>98 THEN ON ERROR GOTO 0
3090 OPEN "I",#NO,"0:ADRES"+STR$(SEQ)
3100 REM*****LOOK FOR THE FINAL FILE*****
3110 ON ERROR GOTO 3180
3120 PREV=NO:SEQ=SEQ+1
3130 NO=NO+1:IF NO>2 THEN NO=1
3140 OPEN "I",#NO,"0:ADRES"+STR$(SEQ)
3150 CLOSE #PREV
3160 GOTO 3120
3170 REM**WHEN YOU RUN OUT OF THE FILE**
3180 IF ERR<>63 THEN ON ERROR GOTO 0
3190 RESUME 3200
3200 REM********* MAIN *********
3210 OPEN "O",#3,"0:ADRES"+STR$(SEQ)
3220 IF EOF (PREV) GOTO 3430
3230 INPUT#PREV NAME $ ,ADDRESS$, TEL$
3240 PRINT "NAME:";NAME$
3250 PRINT "ADDRESS:";ADDRESS$
3260 PRINT "TELEPHONE:"; TEL$
3270 PRINT "1: LEAVE IT AS IS 2: DELETION 3: CORRECTION
3280 INPUT "WHICH CHOICE";A
3290 IF A < 1 OR A > 3 GOTO 3270
```

```
3300 ON A GOTO 3410, 3220, 3320
3310 REM*****CORRECTION*****
3320 PRINT "1:NAME   2: ADDRESS 3: TELEPHONE
3330 INPUT "CORRECT WHICH NO.";B
3340 IF B<1 OR B>3 GOTO 3320
3350 INPUT "ENTER CORRECTION":C$
3360 IF B=1 THEN NAME$ =C$
3370 IF B=2 THEN ADDRESS$=C$
3380 IF B=3 THEN TEL$=C$
3390 PRINT "ALL RIGHT?":GOTO 3240
3400 REM***** CREATE A RECORD *****
3410 PRINT#3, NAME$,CHR$(44),ADDRESS$,CHRS$(44),TEL$
3420 GOTO 3220
3430 CLOSE#PREV
3440 INPUT"ADDITION(Y/N)";A$
3450 IF A$="N" GOTO 3520
3460 IF A$<  "Y" GOTO 3440
3470 INPUT "NAME";NAME$
3480 INPUT "ADDRESS";ADDRESS$
3490 INPUT "TELEPHONE";TEL$
3500  PRINT  3,NAME$, CHR$(44),ADDRESS$,CHR$(44),TEL$
3510 PRINT "MORE";:GOTO 3440
3520 CLOSE# 3
3530 END
```

This program is given with varieties of functions to the extent that no other program is needed once a data file is made, but there are two defective points in it. One is that the program has no function to simply display the whole data, and the other is that all members must be displayed even for one addition. Another program that has these defects will be shown later. At this moment, just follow the explanation to understand flow of modification steps.

The first operation is to find the latest file. This is performed by Lines 3000 through 3190. Line 3010 will be explained later, and Line 3020 performs initialization of file numbers. ⌜NO⌟ is for the file No. and either 1 or 2 is entered. This program is to make a file of "ADDRESS n" and the number that corresponds to n is SEQ. When SEQ is 0, no "n" is attached, and that is the current file "ADDRESS". Line 3030 opens the file.

Since there is a file having the name of ⌜ADDRESS⌟ , execution jumps to Line 3110 by the instruction on Line 3040. Explanation of Line 3110 is also held. Line 3120 enters the file No. that opened ⌜ADDRESS⌟ into ⌜PREV⌟ and updates ⌜SEQ⌟ and ⌜NO⌟ . Since 1 or 2 only is used for ⌜NO⌟ , if the number becomes greater than 2, it is returned to 1 (Line 3130).

Then, Line 3140 opens ⌜ADDRESS 1⌟ (because SEQ is 1). Naturally, there is no such file and ordinarily, it should have caused the ⌜File Not Found⌟ error. The ON ERROR GOTO instruction on Lines 3110 and 3010 prevents the error from occurring. If an error occurs in executing Line 3140, instruction on Line 3110 declared later becomes effective and the operation jumps to Line 3180. Line 3180 checks the error numbers and causes an error should an error of other than File No Found . Line 3190 is an instruction that designates a return address of the main routine after error recovery processing, and in this program, operation jumps to Line 3210. Since the current SEQ is 1, Line 3210 opens ⌜ADDRESS 1⌟ in the output mode.

Next question is what to do for modification next time after the current modification. Since ⌜ADDRESS⌟ and ⌜ADDRESS 1⌟ exist now, a new file of ⌜ADDRESS 2⌟ must be opened. In this case, no error occurs in the preceding Line 3140 and the operation goes to Line 3150. Since File NO. 1 that

opened ⌜ADDRESS⌟ is in PREV, Line 3150 closes PREV. Then the operation returns to Line 3120, saves the current file number and then updates the file number and SEQ tries to open ⌜ADDRESS 2⌟ . If the file has not been prepared, it causes an error and the operation jumps to Line 3180, entering the main routine, and opens ⌜ADRES 2⌟ as a new file.

All in all, on operations of Lines 3110 through 3190, SEQ is updated until the ⌜File Not Found⌟ error occurs, thus obtaining the SEQ value to open a new file.

Explanation thus given assumes presence of a file called ⌜ADDRESS.⌟ What will happen if there is no such file. Processing for such case is covered by Line 3010 and Lines 3050 through 3090. If there is no file named as ⌜ADDRESS⌟, the ⌜File Not Found⌟ error occurs on Line 3030, and operation jumps from Line 3010 to Line 3060. Line 3060 checks the error code, and Line 3080 updates SEQ and NO. No PREV is needed here. Entered in PREV is the file number that has opened the existing file. Line 3090 opens the file of the next SEQ. If such file does not exist, operation of Lines 3060 to 3090, a real file having the smallest SEQ is searched for. When such file is discovered, operation changes to search a file having the last SEQ from Line 3110. If neither ⌜ADDRESS⌟ nor ⌜ADDRESS n⌟ exists, operation enters an indefinite loop, which is checked by Line 3085.

The above is the program flow from Lines 3000 through 3190. What it does is to open a file having the greatest SEQ, enter the file number to PREV and obtain SEQ of a file to be created anew. It is very complicated but read the program and understand the contents well.

As to the main routine starting from Line 3210, a file having new SEQ is opened in the output mode by Line 3210, as explained before. Data is input from the last file by Line 3230. Since ⌜NO⌟ is the file number of the file that could not be opened, the file number that reads in data must be PREV.

Through operation of Lines 3240 to 3290, data is displayed and input for instruction of what to do is received. The operation jumps to a processing routine as input by Line 3300. If ⌜1⌟ is input, operation goes to Line 3410 and prints the data as is. If ⌜2⌟ is input, since 2 means erase, no data is output and the operation jumps to read of the next data from Line 3220. If ⌜3⌟ is input, the operation goes to the correction routine from the next Line 3320.

The correction routine is covered by Lines 3320 through 3390. First, what and how the correction should be made is inquired by Lines 3320 through 3350, the correction is made by Lines 3360 through 3380, and the operation returns to Line 3240 to display the results, and enters the state of waiting for input. If correction is made as expected, designate ⌜1⌟ as is. If there is an input error or correction on another item is needed, designate ⌜3⌟ again.

Data corrected by Line 3410, or data as input, is output, the operation returns to Line 3220 by Line 3420 and waits for data input. For this line number and the second line of Line 3300, Line 3320 must be designated. If 3230 is designated, reading out the data to be read causes an error.

When there is no more data to read in the old file, the operation goes to Line 3430 by Line 3220, which closes the old file and inquiry is made whether or not to add.

If no addition is needed, the operation jumps to Line 3520 and closes the file, terminating the correction routine. Through operation of Lines 3470 to 3490, data to be added to is received, and Line 3500 outputs the added data. When adding completes, the operation jumps to Line 3440 and inquiries if there is further addition.

The key part of this program is the operation of opening the latest existing file, modifying it, and opening of a file to be created anew. Files where the number assigned to the file name is incremented at each file update are called generation files, and this type of file management is called generation management. It is a method frequently used in sequential file management of large computers.

Program 4 is a program which has the function to display the whole data but not to display other data items when modification is only addition of data.

Program 4

```
 10 REM ************************
 20 REM **CATALOG (PROGRAM C)**
 30 REM ************************
 40 REM
 100 REM *****MENU*****
 110 CLS
 120 PRINT TAB (25);"***CATALOG***"
 130 LOCATE 10,3
 140 PRINT "1: DISPLAY ONLY"
 150 LOCATE 10, 6
 160 PRINT "2: ADDITION ONLY "
 170 LOCATE 10,9
 180 PRINT "3: CORRECTION DELETION, ADDITION"
 190 LOCATE 10,12
```

```
200 PRINT "4:END"
210 LOCATE 10,15
220 INPUT "WHICH NO.";A
230 IF A<1 OR A>4 GOTO 220
240 IF A=4 THEN END
250 ON A GOSUB 2000, 1000, 3000
260 GOTO 110
1000 REM***** ADDITION ONLY *****
1010 PRINT "PLEASE WAIT"
1020 GOSUB 4000
1030 OPEN "O",#3, "0:ADDRESS" + STR$(SEQ)
1040 IF EOF (PREV) GOTO 1080
1050 INPUT# PREV, NAME$, ADDRESS$,TEL$
1060 PRINT# 3, NAME$, CHR$(44), ADDRESS$, CHR$(44), TEL$
1070 GOTO 1040
1080 CLOSE# PREV
1090 GOSUB 4600
1100 IF A$="Y" GOTO 1090
1110 CLOSE# 3
1120 RETURN
2000 REM***** DISPLAY ONLY *****
2010 GOSUB 4000
2015 I=1
2020 IF EOF (PREV) GOTO 2050
2030 GOSUB 4500
2032 I=I+1:IF I<7 GOTO 2020
2035 PRINT "PRESS ANY KEY:;
2040 A$=INPUT$(1):PRINT:PRINT:GOTO 2015
2050 CLOSE#PREV
2053 PRINT "IT IS FINISHED. PRESS ANY KEY:;
2060 A$=INPUT$(1):RETURN
3000 REM*****CORRECTION, DELETION, ADDITION
3010 GOSUB 4000
3020 OPEN "O",# 3, "0:ADDRESS"+STR$(SEQ)
```

```
3030 IF EOF(PREV) GOTO 3400
3040 GOSUB 4500
3050 PRINT "1:LEAVE AS IS 2: DELETION 3:CORRECTION";
3060 PRINT "4:ADD IT BEFORE THIS RECORD
3070 INPUT "WHICH NO.";A
3080 IF A<1 OR A>4 GOTO 3050
3090 ON A GOTO 330, 3030, 3110, 3210
3100 REM***** CORRECTION*****
3110 PRINT "1:NAME 2:ADDRESS 3:TELEPHONE"
3120 INPUT "CORRECT WHICH NO.";B
3130 IF B<1 OR B>3 GOTO 3110
3140 INPUT "CORRECTION";C$
3150 IF B=1 THEN NAME$=C$
3160 IF B=2 THEN ADDRESS$=C$
3170 IF B=3 THEN TEL$=C$
3180 PRINT "ALL RIGHT?"
3190 GOSUB 4520:GOTO 3050
3200 REM*****ADD IT BEFORE*****
3210 GOSUB 4600
3220 IF A$="Y" GOTO 3210
3230 GOSUB 4520:GOTO 3050
3290 REM*****WRITE AS IS*****
3300 PRINT 3, NAME$, CHR$(44),ADDRESS$,CHR$(44),TEL$
3310 GOTO 3030
3400 REM**END OF CORRECTION, DELETION AND ADDITION
3410 CLOSE  PREV
3420 GOSUB 4730
3430 IF A$="Y" THEN GOSUB 4600:GOTO 3430
3440 CLOSE#3:RETURN
4000 REM*****OPENING THE LATEST FILE*****
4010 REM
4020 REM***FIND A FILE***
4030 ON ERROR GOTO 4080
4040 PREV=0:SEQ=0:NO=1
```

```
4050 OPEN "I",#1,"0:ADDRESS"
4060 GOTO 4140
4070 REM**WHEN YOU DON'T HAVE AN ADDRESS**
4080 IF ERR<> 63 THEN ON ERROR GOTO (
4090 RESUME 4100
4100 SEQ=SEW+1:NO=NO+1:IF NO>2 THEN NO=1
4110 IF SEQ>98 THEN ON ERROR GOTO 0
4120 OPEN "I",#NO,"( :ADDRESS"+STR$(SEQ)
4130 REM*****FIND THE LAST FILE*****
4140 ON ERROR GOTO 4250
4150 PREV=NO:SEQ=SEQ+L
4160 NO=NO+1:IF NO>2 THEN NO=1
4170 IF SEQ<100 GOTO 4210
4180 CLOSE
4190 PRINT "SEQ HAS OVERFLOWED"
4200 ERROR 21
4210 OPEN "I",#NO,"0:ADDRESS"+STR$(SEQ)
4220 CLOSE# PREV
4230 GOTO 4150
4240 REM** WHEN THE FILE RUNS OUT **
4250 IF ERR<> 63 THEN ON ERROR GOTO 0
4260 RESUME 4270
4270 ON ERROR GOTO 0
4280 RETURN
4500 REM*****READ AND DISPLAY RECORD
4510 INPUT# PREV, NAME$, ADDRESS$,TEL$
4520 PRINT "NAME:";NAME$
4530 PRINT "ADDRESS:";ADRESS$
4540 PRINT "TELEPHONE:";TEL$
4550 PRINT
4560 RETURN
4600 REM****ADDITION OF RECORD****
4610 PRINT "PLEASE INPUT ADDITIONAL DATE."
4620 INPUT "NAME";N$
```

```
4630 INPUT "ADDRESS";N$
4640 INPUT "TELEPHONE";AD$
4650 PRINT "NAME:";N$
4660 PRINT "ADDRESS:";AD$
4670 PRINT "TELEPHONE:";T$
4680 INPUT "IS THIS ALL RIGHT (Y/N) ";A$
4690 IF A$="N" GOTO 4610
4700 IF A$=< >"Y" GOTO 4680
4710 PRINT# 3,N$,CHR$(44),AD$,CHRS$(44),TS
4720 PRINT "FURTHER";
4730 INPUT "ADDITION (Y/N) ";A$
4740 IF A$<>"Y" AND A$<>"N" GOTO 4720
4750 RETURN
```

Explanation of Program 4 is omitted since it is a combination of the preceding programs. Save this program as ⌜ADDRESS 4.⌟

One caution is given for using this program, which is n, (⌜n⌟ of ⌜ADDRESS n⌟), of existing files must be continuous.

For example, if there are files named ⌜ADDRESS 2⌟ , ⌜ADDRESS 3⌟, and ⌜ADDRESS 5⌟ , but no file of ⌜ADDRESS.4⌟ exists as it has been KILLed, the program would recognize the file of ⌜ADDRESS 3⌟ as the latest file. If a file is deleted, all the preceding files must also be deleted.

One more explanation regarding instructions that relate to file handling. Execute the FILES instruction. File of ⌜ADDRESS 1⌟ , ⌜ADDRESS 2⌟ , ⌜ADDRESS 3⌟ , and ⌜ADDRESS 4⌟ will be displayed as program files (files of Kind 0). Files of ⌜ADDRESS 2⌟ and ⌜ADDRESS 3⌟ are no longer needed as long as there are files of ⌜ADDRESS 1⌟ and ⌜ADDRESS 4⌟. Therefore, the two programs can be deleted from the registration, and for this purpose the KILL instruction is used as follows:

```
KILL "0:ADDRESS 2"
KILL "0:ADDRESS 3"
```

Parameter needed for the KILL Instruction is for the file descriptor only.

3.3.4 Summary of Sequential Access File

The following are advantageous points of sequential access files:

(1) Easy programming
(2) Record definition is left with users completely.

On the other hand, they have the following disadvantages:

(1) It requires a long time to find a specific record.
(2) The OPEN and CLOSE instructions must be repeated each time mode for write and read is changed.
(3) Data that has been registered cannot be erased, corrected nor added.
(4) Because of the above, the processing is slow when compared with random access files.

On ordinary file processing, inconvenience caused by these inferior points are more noticeable and users often think that they are of no use. To find out a specific record, files must be read from the start. After finding one record, the file must be CLOSED and OPENed again to find another record. There is a method of reading all data in arrays, but this leaves problems such as what should be the adequate element number of the array, memory capacity, etc., and the method is not realistic.

To improve the third defective point, a file that was made before must be read and a new file must be made while giving due corrections. This procedure, however, may be an

essential point since if an error is made in directly correcting an old file, correct data will have been lost.

It is possible to leave files before correction in the case of random access files, but this is not the way to fully utilize the advantage of random access files. In either case, if leaving files before correction is essential, sequential access files provide ease in programming.

In addition, the record length is fixed in random access files and data items longer than this length must be cut. Very little can be done if the maximum number of items cannot be set. On this point, sequential access files are far superior. Though inconvenient, sequential access files permit realization of all functions.

Nevertheless, for processing simple items like address catalogs, random access files, of which explanation will be given in the next section, seem to be better fitted.

There are other instructions related to handling sequential access files, in addition to those explained so far, but they are scarcely used and explanation is omitted.

3.4 RANDOM ACCESS FILE

3.4.1 Output of Random Access File

This section describes instructions that output random access files.

Program 1 for sequential access files can be rewritten for random access files as shown by Program 5 below.

Program 5

```
1000 OPEN "R",#1,"0: CATALOG"
1010 FIELD #1, 15 AS N$, 40 AS A$,13 AS T$, 2 AS Y$
1020 R=1
1030 INPUT "NAME";NAME$
1040 IF NAME$ ="END"GOTO 1150
1050 INPUT "ADDRESS";ADDRESS$
1060 INPUT "TELEPHONE";TEL$
1070 INPUT "AGE";YEAR
1080 LSET N$=NAME$
1090 LSET A$=ADDRESS$
1100 LSET T$=TEL$
1110 RSET Y$=MKI$(YEAR)
1120 PUT #1,R
1130 R=R+1
1140 GOTO 1030
1150 CLOSE #1
1160 END
```

How does it compare? They look very different from Program 1, but the functions are completely the same. One additional item given in this program is the age, and all others remain same. There are a number of instructions that are different from those of Program 1. Let us go over it line by line.

Line 1000 is the OPEN statement, which is needed in any program. This OPEN statement is the same as that of sequential access files, but the input/output mode is ⌜"R"⌟. In random access files, all of read, addition and correction can be operated with one OPEN-CLOSE.

Since the file name of ⌜ADDRESS⌟ was used in 3.3, a different file name is needed here.

The FIELD Instruction on Line 1010 did not exist in sequential access files. In random files, the record configuration must be clearly defined. Random files provides a new function that was not available with sequential files and that is read and write in units of record, and a number called Record No. is used as the key.

Location of specific records in the disk must be calculated from the record number, and it can be easily understood that size of each record must be fixed to allow the calculation. However, individual data items are not necessarily in the same length. Therefore, how to allocate the excess parts must be specified in advance. The FIELD instruction performs the specification, that is, it determines the file buffer partition.

The first parameter is for file number and designations of data area size and character variables follow to it. ⌜15 AS N$⌟ designates that a space for 15 characters is assigned to the N$ character variable. Rest of the line designates assignment of 40-character space to A$, 13-character space to T$ and 2-character space to Y$. To summarize this, one record is structured as shown in Fig. 3.

N$ Name	A$ Address	T$ Tel No.	Y$ Age
15	40	13	2

Fig. 3 Record Structure

Variables used in the FIELD statement must be character variables. Character variables must be designated for even numerics, which will be explained later. Also be careful that variables used in the FIELD statement must not be used in instruction statements that alter or assign the contents (except LSET and RSET instructions).

Line 1020 initializes the record number.

On Lines 1030 through 1070, data input is received. A strange instruction appears on Line 1080. The LSET instruction is akin to the LET statement. Data is entered in a file buffer by the LSET statment. LSET or RSET is used depending how the data is to be entered. LSET is for left justification and RSET for right justification. Suggested use of these instructions is LSET for name and character and RSET for money amount and age.

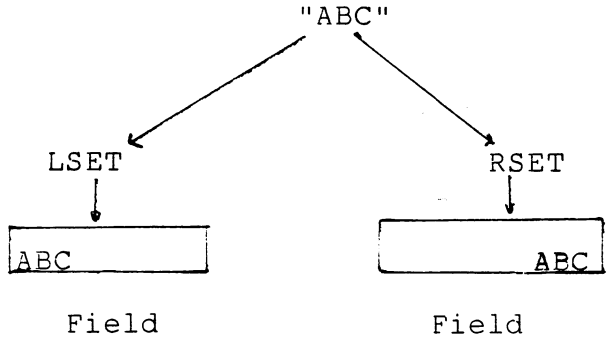


Fig. 4 LSET and RSET.

By the way, you may think that receiving input with the INPUT statement and assigning the input to the buffer with LSET or RSET are double action, but this is inevitable. Variable names defined in the FIELD statement cannot be used in the input statement and data cannot be entered in the buffer by defining with the FIELD statement only, and without LSET or RSET statement.

Lines 1090 and 1100 are both LSET statements.
Line 1110 is RSET statement. MKI$ is a function and it converts numeric data to character type data.

Data registered in random files must be character type data always. Therefore, a function that converts numeric data to character data is needed.

There is another function, STR$, that converts numerics to characters for PRINT statement. The difference between STR$ and MKI$ is that STR$ converts numerics to ASCII codes of decimal notation where-as MKI$ converts data type of internal expression only without changing the contents. For example, 20 in decimal notation is 14 in hexadecimal notation and if it is converted by STR$, it becomes character strings of 32, 30 (hexadecimal in both cases) of ASCII codes. If it is converted by MKI$, the contents of data, 14, remains as is and what is changed is only the index that indicates the data type.

Also note that there are differences in the functions when numerics are converted to characters for random files, depending on if it is integer, single precision real number, or double precision real number. MKI$ is a function used for integers and MKS$ is used for single precision real numbers and MKD$ for double precision real numbers. By these functions, an integer is converted to character data of 2 bytes, single precision real number to that of 4 bytes and double precision real number to that of 8 bytes.

Character strings converted by these functions can be converted back to the original numerics by CVI, CVS and CVD functions respectively, which will be explained in the next section.

PUT on Line 1120 is an output instruction. ⌜# 1⌟ is for file number, ⌜R⌟ for record number and where the data is written is determined by these.

Line 1130 updates the record number.

Line 1140 sets the operation to data input again.

Since correspondence between variables, and data is defined in the FIELD statement in random files, delimit signs as used in sequential files are not needed.

Line 1150 closes the file. Function of the CLOSE instruction is same to both sequential and random files.

Program 5 can be used in the same way as Program 1, but there is no need of preparing the same file as was prepared in Section 3.3. Program 6 is a program that converts the sequential file created in Section 3.3 to a random file. Beware that use of Program 6 after using Program 5 will erase the file created by Program 5 completely since the file to be created bears the same file name. Or, change the file name of Program5 appearing on Line 1000. Do not change the file name of Program6, as otherwise all the following programs in this manual must be changed. If a ⌜catalog⌟ file has already been made, change the file name using the NAME instruction.

Program 6

```
10 OPEN "I",#1,"0:ADDRESS"
20 R=1:OPEN "R",#2, "0: CATALOG"
30 FIELD#2,15 AS N$, 40 AS A$,13 AS T$,2 AS Y$
40 IF EOF (1) GOTO 140
50 INPUT#1, NAME, ADDRESS$, TEL$
60 PRINT NAME$:";ADDRES$
65 PRINT TAB(10)
70 INPUT "AGE";YEAR
80 LSET N$+NAME$
90 LSET A$=ADDRESS$
100 LSET T$=TEL$
110 RSET Y$=MKI$(YEAR)
120 PUT#2,R
130 R=R+1:PRINT:GOTO 40
140 CLOSE#1,#2
150 END
```

Lines 10 and 20 open the files, and the sequential file is numbered as 1 and the random file as 2. Therefore, the file number on Line 30 must be 2. Line 20 initializes the record number also.

Line 40 determines data end and Line 50 reads data. Since the sequential file does not contain ages, after displaying names and addresses by Line 50 , key in is received by Line 70.

Sample run 5

```
RUN
JOHN SMITH    15 WEST ST. MELBOURNE
              AGE? 27
JOE BROWN     1  NORTH ST. ADELAIDE
              AGE? 23
FRED WHITE    222 SOUTH ROAD. SYDNEY
              AGE? 32
```

Lines 80 through 110 set the data to the buffer and Line 120 writes the data.

Line 130 updates the record number and operation returns to data input again.

When there is no data to input, the operation terminates closing the file with Line 140. When the file number in the CLOSE statement is omitted,all files in open state are closed.

3.4.2 Input of Random Access File

Program 7 is an input program of random access files.

Program 7

```
2000 R=0:CLS
2010 OPEN "R",#1,"0: CATALOG"
2020 FIELD#1,15 AS N$,40 AS A$,13 AS T$,2 AS Y$
2030 PRINT TAB(25);"*** CATALOG ***"
2040 PRINT "NAME";TAB(18);"ADDRESS"
2045 PRINT TAB(63);"TELEPHONE;TAB(74); "AGE"
2050 FOR I=1 TO 22
2060 IF R> = LOF(1) THEN I=24:GOTO 2090
2070 R=R+1:GET#1,R
2080 PRINT N$;TAB(15);AS;TAB(60);T$;TAB(75);CVI(Y$)
2090 NEXT I
2100 IF I>23 GOTO 2120
2110 PRINT "PLEASE PUSH ANY KEY";:B$=INPUT$(1)
2115 PRINT:GOTO 2030
2120 CLOSE# 1
2130 END
```

Line 2000 is for initialization of the record number. Line 2010 opens the file. Line 2020 sets the FIELD statement, which is needed for programs to only read files.

Lines 2030 and 2040 write the heading and Lines 2050 through 2090 read and display the data, and processing here is rather complicated.

Sample run 6

NAME	ADDRESS	***CATALOG***	TELEPHONE	AGE
JOHN SMITH	15 WEST ST.	MELBOURNE	(03)062-4952	22
JOE BROWN	1 NORTH ST.	ADELAIDE	(08)098-7654	23
FRED WHITE	222 SOUTH RD.	SYDNEY	(02)054-2931	32

25 lines can be displayed on the screen, but since 2 lines are used for the heading and 1 line at the end for waiting for input, when data for 22 lines is displayed, the display stops and the system waits for input. Only one line is used for one record.

There are 2 purposes for designating 24 for I on Line 2060 and that is, to terminate the FOR-NEXT loop when the data ends and to know if the data ended when operation comes out of the FOR-NEXT loop. If data has not ended by Line 2100, the FOR-NEXT loop is executed in the state of I being 22, and then I becomes 23 by NEXT I on Line 2060 and the operation jumps to Line 2090 and I becomes 25 by ⌜NEXT I⌟ . Therefore, if I is greater than 23, it means that there is no more data.

Line 2080 displays data, but as you remember, in the preceding section it was said that variable names defined in the FIELD statement cannot be used in the INPUT statement.

This is because statement like PUT and GET do not function properly with variables defined in the FIELD statement unless LSET or RSET is used to define or alter the contents but in the case of the PRINT Instruction, since it is not an instruction that changes the contents, use of these variables does not bring an inconvenience.

The function of CVI appears here as predicted in the preceding section. At the time MKI$ was used and the following is a review of the function.

(1) Data to be stored in random file buffers by LSET or RSET must be in character type.
(2) MKI$ is used to convert integer data to character type, MKS$ for single precision real number data and MKD$ for double precision real number data.

The following summarizes conditions for CVI:

(1) If data read into a random file buffer by the GET statement was originally a numeric; CVI,CVS or CVD is used to convert it back to the original numeric data.

(2) If the original data was an integer, use CVI to convert the character data to the orignal numeric. In the same way, use CVS In the case of single precision real number data and CVD in the case of double precision real number data.

This explanation can be rephrased as follows.

(2) Data converted by MKI$, data converted by MKS$ and data converted by MKD$ can be converted back to the original type by using CVI, CVS and CVD respectively. I stands for Integer, S for Single and D for Double.

If data has not ended, the system waits for input as stated in Line 2110. If the [RETURN] key is pressed, the operation

returns to displaying the heading as programmed on Line 2030.

If data has ended, the file is closed and the operation terminates.

A point to be noted in the FIELD statement at the time of input is that the number of characters of each data item must concur with that of output program. The variable names need not be the same. Other than these points, the input program is simpler than the output program.

The next explanation on read/write in units of record, which is one of the characteristics of random access files.

3.4.3 Modification of Random Access File

The section describes modification of random access files which involves addition, correction and deletion of data.

With sequential access files, addition, correction and deletion of data cannot be made directly to the files, and what has to be done is to read the file before modification, give necessary modification after reading a specific record and put it back to the same file. To instruct the computer which record to be read and where to store the modified record, record numbers are used.

To add data, designate the next number of the last record of the current file as the record number. To correct or delete data, designate the record number to which the processing is needed, and designate the same record number as before.

Doing all these at a time complicates the program. Therefore, the first example, Program 8, is for adding data.

Program 8

```
3000 OPEN "R",#1,"CATALOG"
3010 FIELD#1,15 AS N$,40 AS A$,13 AS T$,2 AS Y$
3020 R=LOF(1)
3030 INPUT "NAME";NAME$
3040 IF NAME ="END" GOTO 3120
3050 INPUT "ADDRESS";ADDRESS$
3060 INPUT "TELEPHONE";TEL$
3070 INPUT "AGE";YEAR
3080 LSET N$=NAME$:LSET A$=ADDRESS$
3090 LSET T$=TEL$:RSET Y$=MKI$(YEAR)
3100 R=R+1:PUT #1,R
3110 PRINT:GOTO 3030
3120 CLOSE#1:END
```

Other than Line 3020, Program 8 is completely the same as Program 5, an output program, that is initialization of the record number is the only different point.

The function of LOF, which by the way is closer to a system variable rather than being a function, is set by the OPEN instruction. Give the largest number of the record numbers in the designated file. This does not necessarily mean the number of record that was written latest, as records having larger number can be written in before younger number records are written. When this feature is utilized, record management like assigning record numbers up to 50 to names starting with A, from 51 to 100 to names starting with B, and from 101 up to 150 to names starting with C, etc However, if the record system is made in such way, the program to be prepared for adding data cannot be so simple as Program 7.

There is another function that relates to record numbers, and that is LOC. The name is similar to LOF, so is the

meaning, and care must be paid not to mix the two.

For the function of LOC, give the next number of the record that was accessed latest. In the GET and PUT statements, the record number can be omitted, and if it is omitted, the result is the same as designating the next number of the record that was accessed latest. In other words, for sequential access like Programs 5, 6 and 7, there is no need of assigning a record number. (However, be sure to understand that in read/write operation whcih is explained from now on, access is not necessarily in the sequence of record numbers, and record numbers must be designated.) LOC is a record number that is used when the record number is omitted.

Once again, we call your attention not to mix LOF and LOC as they are spelled very similar.

The next explanation is on a program for correction and deletion in units of record. See Program 9.

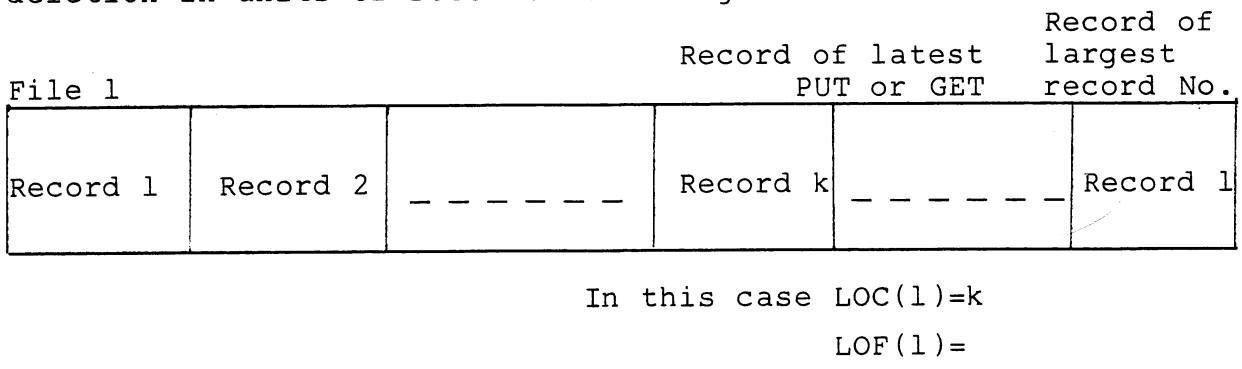


Fig.5 LOF and LOC

Program 9

```
3500 OPEN "R",#1,"CATALOG"
3510 FIELD#1,15 AS N$,40 AS A$,13 AS T$,2 AS Y$
3520 INPUT "RECORD NO.";R:IF R<=LOF(1) GOTO 3530

3525 PRINT "THE NO. OF RECORDS IS:";LOF(1);GOTO 3520
```

```
3530 GET # 1,R
3540 PRINT "NAME";N$;TAB(25);"ADDRESS:";A$;TAB(20);
3550 PRINT "TELEPHONE";T$;TAB(43);"AGE:";CVI(Y$)
3560 INPUT "1:CORRECTION  2: DELETION  3: DO NOTHING";B
3570 IF B<1 OR B>3 GOTO 3560
3580 IF B=3 GOTO 3600
3590 ON B GOSUB  3700,3900
3600 PUT# 1,R
3610 PRINT "NEXT (Y/N)?";
3620 B$=INKEY$:IF B$="Y" THEN PRINT:GOTO 3250
3630 IF B$ <> "N" GOTO 3620
3640 CLOSE# 1:END
3700 REM **** CORRECTION ****
3710 PRINT TAB(5);"1:NAME 2:ADDRESS 3:TELEPHONE";
3720 PRINT TAB(39);"4 AGE "
3730 B$=INKEY$:IF B$="" GOTO 3730
3735 IFB$<>"1"ANDB$<>"2"ANDB$<>"3"ANDB$<>"4" GOTO 3730
3740 PRINT:PRINT "CHANGE TO";
3750 IF B$="4" THEN INPUT C ELSE INPUT C$
3760 IF B$="1" THEN LSET N$=C$
3770 IF B$="2" THEN LSET A$=C$
3780 IF B$="3" THEN LSET T$=C$
3790 IF B$="4" THEN RSET Y$=MKI$(C)
3800 PRINT "NAME:";N$;"ADDRESS:";A$
3810 PRINT TAB(20);"TELEPHONE:";T$;"AGE:";CVI(Y$)
3820 PRINT "MORE CORRECTIONS(Y/N)";
3830 B$-INKEY$:IF B$="Y" THEN PRINT:GOTO 3710
3840 IF B$<>"N" GOTO 3830
3850 PRINT:RETURN
3900 REM ***DELETION***
3910 PRINT "DELETE RECORD";N$;"?(Y/N)";
3920  B$=INKEY$:IF B$="N" THEN PRINT:GOTO 3560
3930 IF B$< > "Y" GOTO 3920
3940 LSET N$="":LSET A$="":LSET T$="":RSET Y$=""
3950 PRINT:RETURN
```

It is a long program. Lines 3500,3510 and 3520 are to OPEN the file, definition of FIELD and designation of the record number, respectively. To designate the record to correct or delete with the name is more convenient, but to do so, another file must be prepared for a table showing correspondence of names and record numbers which complicates the program. Therefore, right now, record number of the objective person is searched from display of the total members. Program 7 is not made to display record numbers, but record numbers are displayed when the program is completed.

Lines 3530 through 3550 read and display the record having the designated record number. Line 3560 receives selection of whether to correct or to delete. Do nothing is provided for the case that an error is made on the record number and the read record is not the objective record.

Line 3570 checks if the input is correct. If Do Nothing was selected, the operation jumps from Line 3580 to Line 3600. In Line 3590, operation jumps to a subroutine of correction or deletion.

Sample run 7

```
RUN
RECORD NO.?1
NAME: JOHN SMITH  ADDRESS:  15 WEST ST. MELBOURNE
       TELEPHONE:(03)062-4952    AGE:27
1: CORRECTION 2:DELETION 3:DO NOTHING  ?1
    1:NAME 2: ADDRESS 3:TELEPHONE 4: AGE
    4
    CHANGE TO? 22
NAME: JOHN SMITH  ADDRESS: 15 WEST ST. MELBOURNE
TELEPHONE:  (03)062-4952  AGE: 22
MORE CORRECTIONS (Y/N)  N
NEXT (Y/N)? N

Ready
```

When all processings complete, the operation returns to Line 3600 and outputs the record. Then, whether or not another record would be modified is inquired by Lines 3610 to 3620, and if modification is not longer needed, Line 3640 closes the file and operation terminates as soon as E is input.

Operation from Line 3700 on is a subroutine for correction. Lines 3710 through 3730 display the objective record and inquire what to correct. Lines 3740 through 3750 inquire how to change. Lines 3760 through 3790 change the record contents and Lines 3800 through 3810 display the changed results.

Lines 3820 through 3830 inquire if to correct other items and if there is no data to correct, the operation returns.

Lines 3820 through 3830 inquire if to correct other items and if there is no data to correct, the operation returns.

Line 3900 and on are for a deletion subroutine. Lines 3910 through 3920 checks if the record can surely be deleted, and Line 3930 deletes the contents.

Since there are many similar processes, the program has become lengthy when compared with previous programs, but the file operation is not too complicated as there is no new instructions.

3.4.4 Summary of Random Access File

Program 10 is an address catalog program that combines all the preceding programs.

Program 10

```
100 REM ***** CATALOG *****
110 OPEN "R",#1,"0:CATALOG"
120 FIELD #1,15 AS N$, 40 AS A$,13 AS T$,2 AS Y$
130 LOR=LOF(1)
140 CLS
150 PRINT TAB (3);"***CATALOG***"
```

```
160 LOCATE 10,3:PRINT "1.DISPLAY OF ALL RECORDS"
170 LOCATE 10,5:PRINT "2.DISPLAY OF SPECIFIC RECORDS"
180 LOCATE 10,7:PRINT "3.ADDITION"
190 LOCATE 10,9:PRINT "4.CORRECTION"
200 LOCATE 10,11:PRINT "5.DELETION"
210 LOCATE 10,13:PRINT "6.END"
220 LOCATE 1,15:PRINT "WHICH NO.?"
230 B$+INKEY$:B=VAL(B$):IF B<1 OR B>6 GOTO 230
235 PRINT TAB(10);CHR$(30);B$:PRINT
240 ON B GOTO 1000,3000,2000,3000,3000,250
250 CLOSE#1
260 END
1000 REM *****DISPLAY ALL*****
1010 R=0
1020 CLS
1030 PRINT TAB(30);"***CATALOG***"
1040 PRINT"NO.NAME";TAB(23);"ADDRESS";
1045 PRINT TAB(63);"TELEPHONE;TAB(75);"AGE"
1050 FOR I=1 TO 22
1060 IF R>=LOF(1) THEN I=24:GOTO 1090
1070 R=R+L:GET# 1,R
1075 IF N$="                    "GOTO 1060
1080 PRINT R;TAB(5);N$;TAB(20);A$;
1085 PRINT TAB(60);T$;TAB(74);CVI(Y$)
1090 NEXT I
1100 IF I>23 GOTO 1130
1110 PRINT "PRESS ANY KEY TO CONTINUE";
1120 B$=INKEY$:IF B$="" GOTO 1120 ELSE GOTO 1020
1130 PRINT
1140 PRINT "FINISH.PRESS ANY KEY TO RETURN TO MENU";
1150 B$=INKEY$:IF B$="" GOTO 1150 ELSE GOTO 140
2000 REM ****ADDITION OF RECORD*****
2010 LOR=LOR+1
2020 INPUT "NAME"NAME$
```

```
2030 INPUT "ADDRESS";ADDRESS$
2040 INPUT "TELEPHONE";TEL$
2050 INPUT "AGE";YEAR
2060 LSET N$=NAME$:LSET A$=ADDRESS$
2065 LSET T$=TEL$:RSET Y$=MKI$(YEAR)
2070 GOSUB 4000
2080 PRINT "ALL RIGHT? (Y/N)";:GOSUB 4100
2085 PRINT TAB(20);CHR$(30);B$
2090 IF B$="N" THEN R=LOR:GOTO 3210
2100 PUT#1,LOR
2110 GOSUB 4100
2130 IF B$="N" GOTO 140
2135 PRINT TAB(20);CHR$(30);B$
2140 PRINT:GOTO 2010
3000 REM ***** READING A RECORD *****
3010 INPUT "RECORD NO";R:IF R>LOR GOTO 3015
3013 IF R<1 GOTO 3010 ELSE GOTO 3020
3015 PRINT "THE RECORD NO.IS UP TO";LOR:GOTO 3010
3020 GET#1,R
3025 IF N$="                    "GOTO 3070
3030 GOSUB 4000
3040 ON B GOTO 140,3050,140,3200,3100
3050 PRINT "PRESS ANY KEY TO RETURN TO MENU"
3060 B$=INKEY$:IF B$="" GOTO 3060 ELSE GOTO 140
3070 PRINT "RECORD NO.";R;"IS NOT REGISTERED"
3080 GOTO 3050
3100 REM*****DELETION*****
3110 PRINT"DELETE THIS RECORD(Y/N)"
3120 GOSUB 4100
3130 IF B$="N" GOTO 140
3140 LSET N$="":LSET A$="":LSET T$="":RSET Y$=""
3150 PUT#1,R
3160 GOTO 140
```

```
3200 REM *****CORRECTION*****
3210 PRINT TAB (5);"1:NAME":
3220 PRINT TAB(15);"2:ADDRESS";
3230 PRINT TAB (25);"3:TELEPHONE";
3240 PRINT TAB(35);"4:AGE";
3245 PRINT TAB(45);"5:RETURN TO MENU"
3250 PRINT "CORRECT WHICH "(No)?
3260 B$=INKEY$:B=VAL(B$):IF B<1 OR B>5 GOTO 3260
3263 IF B=5 GOTO 140
3265 PRINT TAB(15);CHR$(30);B$:PRINT
3270 INPUT "CHANGE TO";C$
3280 IF B=1 THEN LSET N$=C$
3290 IF B=2 THEN LSET A$=C$
3300 IF B=3 THEN LSET T$=C$
3310 IF B=4 THEN REST Y$=MKI$(VAL(C$))
3320 GOSUB 4000
3330 PRINT "CORRECT OTHER ITEMS (Y/N)"
3340 GOSUB 4100
3350 IF B$="Y" GOTO 3210
3360 PUT#1,R
3370 GOTO 140
4000 REM *****DISPLAY OF A RECORD*****
4005 PRINT
4010 PRINT TAB(5);"NAME:";N$;TAB(30);"ADDRESS:";A$
4015 PRINT TAB(5);TELEPHONE:";T$;
4020 PRINT TAB(30);"AGE:";CVI(Y$)
4030 PRINT:RETURN
4100 REM *****Y OR N KEY INPUT*****
4110 B$=INKEY$:IF B$<>"N" AND B$<>"Y" GOTO 4110
4120 PRINT:RETURN
```

It sure is a long program, However, it is nothing but a combination of all the preceding programs.

The main points common with other programs are processings are executed in the main routine only, read operation for display, correction and deletion is placed in one place (separate places when all records are to be displayed only), and all similar routines are placed in one place as a sub-routine. These could be realized because OPEN and CLOSE are needed only once.

The first operation is the main routine in Lines 100 through 260. It basically is the same as the main routine that controls input/output to and from sequential files, Program 3. Differences are as follows:

(1) OPEN and CLOSE were executed for each subroutine in sequential files since input and output modes had to be set. In random files, however, no mode change is needed. so that the two are executed only once in the main routine.

(2) Program 3 is for registration and display only, but Program 10 includes operation of addition, correction, deletion and display of one record as well.

The following describes each step. Line 110 is for OPEN and Line 120 sets the FIELD statement. Line 130 reads the largest record number at the time of OPEN into the LOR variable. The variable is used for adding of data to records.

The main loop starts from Line 140, and lines up to 220 draw a menu screen. Each processing is executed if the number shown in the menu screen is designated. Line 230 receives the key input and checks the number concurrently.

Line 240 instructs to go to the processing routine as input. The display routine of a specific record starting from Line 3000 is also a display routine of the correction

and deletion to be read out.

When the processing completes and END of 6 is designated, the file is closed by Lines 250 through 260 and the operation ends.

Lines for 1000 through 1120 are for the routine to display records of all members and the lines are used to learn the record number of a specific person also in addition to the use of seeing data of all members, since the record number must be designated for correction or deletion.

Line 1010 is for initialization of the record number. Line 1020 is for clearing of the screen and Lines 1030 and 1040 write the heading.

The FOR-NEXT Loop on Lines 1050 through 1090 is for display of data for one screen, to wait for key input and then to read the next data. In one screen, data for 22 lines can be displayed, deducting 2 lines for the heading and one line at the bottom from the total 25 lines.

What is described in Line 1060 is that Loop counter I is made greater than usual when data end is detected so that determination of data end is possible even after coming out of the FOR-NEXT loop. This processing is completely the same as Program 7.

Line 1070 updates the record number and reads the data. Line 1080 displays the data.

Line 1100 is for determination of data ends. If data has ended, the operation returns to display of the menu screen from Line 140.

key input is waited for by the Lines 1110 through 1120 and the operation returns to Line 1020 where data is read in again.

Lines 2000 through 2140 are for adding of data. If data is added, the maximum record number increases and Line

2010 updates LOR. Variable LOR is updated when data is added only, and not used in other processings.

Data input is received by Lines 2020 through 2050, sent to the buffer by Line 2060, and checked if the key input was correct by Lines 2070 through 2090. The reason for checking data that has been sent to the buffer is that sometimes size of the input data is too large, overflowing from the field assigned to the buffer. If there is an error in the input data, instead of receiving the data again, the operation goes to the routine of correction. In the correction routine, ⌜R⌟ is used as the record number, and variable LOR that indicates the record number being used here is assigned to R . The subroutine in Line 4000 is for display of the file buffer contents. The subroutine in Line 4100 is for wait of key input and it does not accept input other than ⌜Y⌟ and ⌜N⌟.

If there is no error in the data input, the data is output (Line 2100).

Lines 2110 through 2130 inquire if there is more data to be added. If there is, the operation returns to Line 2010 and input is received. If there is no more data to be added, the operation returns to the menu screen.

Lines 3000 through 3030 are for display of data in one record, and record of the designated record number is read and displayed. This operation is for additional purpose to check the data before modification, after completing the correction or deletion.

The operation jumps to individual processing routine by Line 3040. Any processing that does not require this routine, like ⌜1: Display of all records⌟ or ⌜3: Adding of record⌟ , is returned to the main loop. The operation goes to Line 3200 if for correction, or Line 3100 if for

deletion. If display is the only operation required, input of any key is waited for by Line 3060 and the operation returns to the menu screen.

Lines 3100 through 3160 are for deletion processing. Whether or not the data can really be deleted is checked, in the same as in Program 9 first, and the buffer is emptied and written. When this processing ends, the operation returns to the menu screen.

Lines 3200 through 3370 are for correction routine. If an input error is made in adding the record, the operation jumps to this routine also. Data to be corrected is designated with the number in Lines 3200 through 3260. Input of new data is received by Line 3270, and the new data is written in the designated data field by Lines 3280 through 3310. Carefully note Line 3310.

Line 3320 displays the corrected results, and Lines 3330 through 3350 inquire if there is another correction to be made. If there is, the operation returns to the start of this routine, and if not, data is output by Line 3360 and the operation returns to the menu screen.

Lines 4000 through 4030 are for the subroutine that displays the data buffer contents. Pay attention to the Y$ processing of Line 4020.

Lines 4100 through 4120 are for operation of checking key in, and only when key in of either ⌜ Y ⌟ or ⌜ N ⌟ , does the operation returns.

Input of other keys are completely ignored.
The program itself is very long but since all processing is rather simple, we believe that readers have understood the program now.

At the end, features of random access files are summarized. They are repeats of the same points as explained in the sequential file section, but in reverse expression.

Random access files have the following advantageous points:

(1) OPEN and CLOSE need not be repeated for each mode of input and output.

(2) Data can be read out at high speed in units of record regardless of the location in the disk.

(3) Correction and deletion in units of record are possible.

Random access fields have the following disadvantages.

(1) Programs required for random access files are very complicated.

(2) There is no flexibility in the size of individual data item. A part of data must be truncated if its size is longer than the predetermined size.

(3) Data is in a fixed length and written in many places of the disk, and efficient use of the disk is poor.

On the data length and record length referred to in Item (2), there are many restrictions imposed on them, and there are the following restrictions in addition to what has been explained:

(1) Length of one record must not exceed 128 bytes (128 characters).
(2) Number of records per file must not exceed 624.

Also, there are the following two restrictions, which are general rules to be applied to DISK BASIC, and not limited to random access file:

(1) Number of files to be written in one diskette is limited to 40.

(2) Number of files that can be opened simultaneously is limited to 16.

Since there are so many restrictions, readers may wonder why there are so many, but further explanation is omitted as it requires too many pages and the explanation involves very difficult subjects. The only restriction that causes inconvenience, is not to exceed 128 bytes.

The third disadvantage may be difficult for readers to understand from explanations that have been given so far. In the case of sequential files, only output is made at the time of output, so that data can be written in continuously from the point where write started. In the case of random files, however, since the record length is fixed, an area of one sector (=128 bytes) is needed to record a small data item of 50 bytes (one sector is preserved as an area for one record always). If the same area is written with a sequential file, data that is equivalent to slightly over 2-record size can be written.

Also, since individual data items are in a fixed length, it often occurs that only about half of the area is used on some data items, but the area is not enough for other items, causing a necessity of abbreviating the addresses.

These are sacrifices needed for the sake of obtaining high speed and flexible processing.

Access to disks without file operation is explained in the last section.

3.5 OTHER DISK OPERATIONS

3.5.1 DSKI$

DSKI$ is a function that returns contents of a certain diskette position and it resembles the PEEK function that returns the memory contents. One difference is that the PEEK function returns data of one byte and the DSKI$ function returns the diskette contents as character strings of 128 bytes. Also, the PEEK function designates the position of data to be read out by address designation, where as the DSKI$ function designates the data position with a track number and sector number in addition to a drive number.

Data is written on diskettes concentrically by a magnet Each of the concentric circle is called a track. The tracks are numbered sequentially from outside. Each track is divided into 16 sectors and one sector consists of 128 bytes. Remember the DSKF function explained in 3.2. DSKF defines empty areas of disks in units or record and one group consists of 4 sectors.

The DSKI$ function returns data of 1 sector (=128 bytes) with the track number and sector number being designated.

This function is useful to know format and structure of files stored in disks and how sequential and random files are being stored. Much knowledge and effort are required to obtain such information by actually reading disk contents.

A sample program for hexadecimal dump of a drive, track and sector designated by key in is shown in Program 11. No instruction of OPEN or CLOSE nor FIELD statement is needed to use the DSKI$ function.

In the character part, a period is used for all control codes. Explanation of this program is omitted.

DISKCOPY explained in 1.3 UTILITY also uses the DSKI$ function. Read the part to make sure you understand the function.

Program 11

```
10 INPUT "DRIVE";D
20 IF D<0 OR D>1 GOTO 10
30 INPUT "TRACK";T
40 IF T<0 OR T>39 GOTO 30
50 INPUT "SECTOR";S
60 IF S<1 OR S>16 GOTO 50
70 A$=DSKI$(D,T,S)
80 FOR I=1 TO 16
90 B$=""
100 FOR J=1 TO 8
110 C$=MID$(A$,(I-1)*8+J,1)
120 A=ASC(C$)
130 D$=HEX$(A)+" "
140 IF LEN(D$)=2 THEN D$="0"+D$
150 PRINT D$;
160 IF A<31 OR A=255 THEN C$="."
170 B$=B$+C$
180 NEXT J
190 PRINT "  ";B$
200 NEXT I
210 END
```

Sample run 8

```
RUN
DRIVE  ?0
TRACK  ?4
SECTOR ?13
```

```
EB  E0  E1  20  E0 FB  93  20    ひだりかっこ
20  20  20  20  20 20  20  E4
93  97  8E  93  E4 20  F0  E5    うさうとかな
E4  98  20  E6  9C 9C  FD  EA    よくにしししは
DE  9C  20  31  35 2D  36  2D    "し 15-6-
32  34  20  20  20 20  20  20    24
20  20  20  20  20 02  20  28              (
30  33  29  30  36 32  2D  34      03)062-4
39  35  32  20  00 16  00  00     952 .....
00  00  00  00  00 00  00  00     .........
00  00  00  00  00 00  00  00     .........
00  00  00  00  00 00  00  00     .........
00  00  00  00  00 00  00  00     .........
00  00  00  00  00 00  00  00     .........
00  00  00  00  00 00  00  00     .........
00  00  00  00  00 00  00  00     .........

Ready
```

3.5.2 DSKO$

Readers may think that DSKO$ is a function, but it is not a function but instruction. DSKI$ is a function to read disk contents whereas DSKO$ is an instruction that directly writes into disks. The relation is similar to that between PEEK and POKE.

It was mentioned that careless use of the POKE instruction would destroy BASIC programs. The same, or even more careful attention is needed when using DSKO$, because, in the case of POKE, what is destroyed is a program made by you and the BASIC interpreter contained in ROM is not affected. However, erroneous use of DSKO$ sometimes may destroy the DISK BASIC itself. For this reason, use of the DSKO$ instruction is not recommended unless it is absolutely necessary.

For a sample program, refer to DISKCOPY IN 1.3 UTILITY.

3.5.3 SAMPLE RUNS OF PROGRAM 11.

Immediately after DSKINI

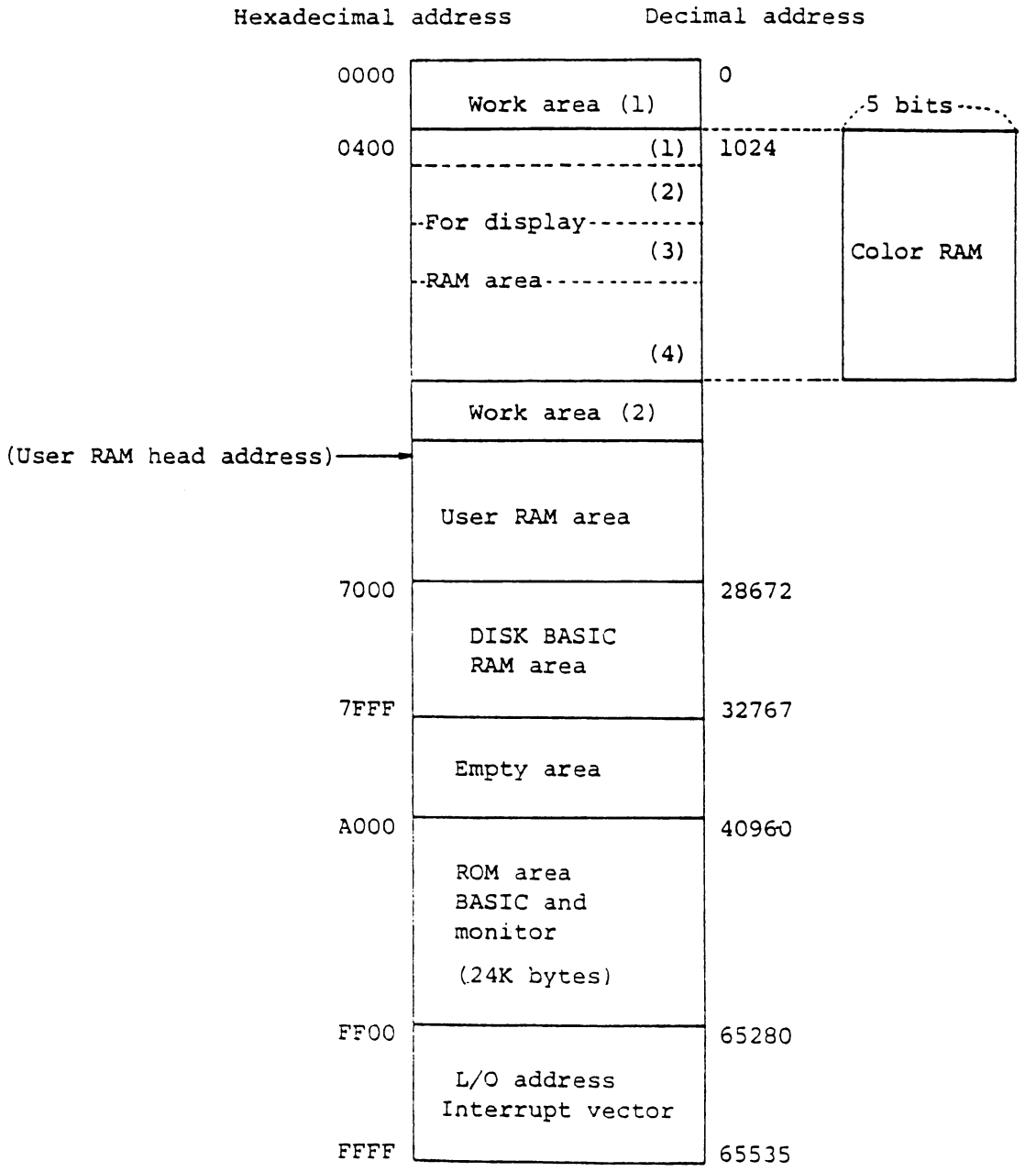
```
DRIVE   ?1
TRACK   ?20
SECTOR  ?1

          00   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........

DRIVE ?1
TRACK ?20
SECTOR ?7

          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
          FF   FF   FF   FF   FF   FF   FF   FF    ........
```

When File is recorded.

```
FILES"1:"
ADDRESS    1 A S 6
Ready

DRIVE      ?1
TRACK      ?20
SECTOR     ?1

00   FF   FF   FF   FF   01   02   03   ........
04   05   C1   FF   FF   FF   FF   FF   ..ﾁ.....
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........

DRIVE      ?1
TRACK      ?20
SECTOR     ?7

41   44   44   52   45   53   53   20   ADDRESS
00   00   00   01   FF   00   00   00   ........
00   00   00   00   00   00   00   00   ........
00   00   00   00   00   00   00   00   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
FF   FF   FF   FF   FF   FF   FF   FF   ........
```

4. REFERENCE MATERIALS

4.1 MEMORY MAP (with standard installation of RAM 32K bytes)



Configuration of RAM area for display and user RAM area

Area	Display mode	Number of display area bytes	User RAM head address, hexadecimal (Figures in parentheses are decimal addresses.)
(1)	40-character, normal	1K bytes	0F72 (3954)
(1) ∿ (2)	80-character, normal	2K bytes	1372 (4978)
(1) ∿ (3)	40-character, high resolution	8K bytes	2B72 (11122)
(1) ∿ (4)	80-character, high resolution	16K bytes	4B72 (19314)

4.2 CHARACTER CODE TABLE

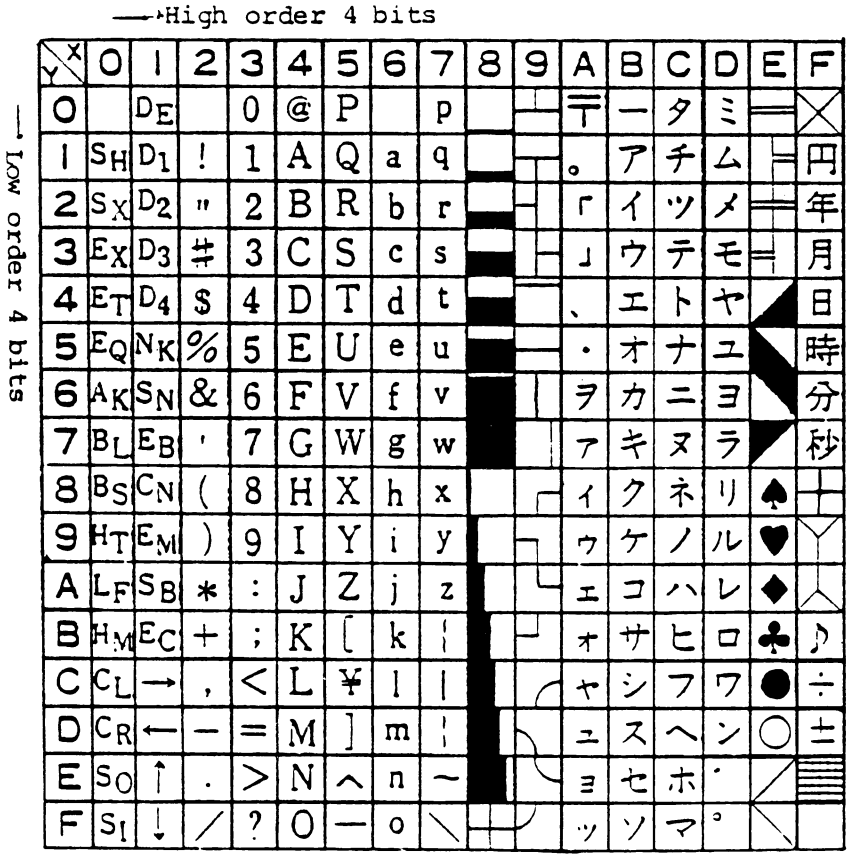
4.2.1 In Interlace Mode

→ High order 4 bits

↓ Low order 4 bits

X / Y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		D_E		0	@	P		p	年	π	〒	ー	タ	ミ	た	み
1	S_H	D_1	!	1	A	Q	a	q	月	あ	。	ア	チ	ム	ち	む
2	S_X	D_2	"	2	B	R	b	r	日	い	「	イ	ツ	メ	つ	め
3	E_X	D_3	#	3	C	S	c	s	市	う	」	ウ	テ	モ	て	も
4	E_T	D_4	$	4	D	T	d	t	区	え	、	エ	ト	ヤ	と	や
5	E_Q	N_K	%	5	E	U	e	u	町	お	・	オ	ナ	ユ	な	ゆ
6	A_K	S_N	&	6	F	V	f	v	を	か	ヲ	カ	ニ	ヨ	に	よ
7	B_L	E_B	'	7	G	W	g	w	ぁ	き	ァ	キ	ヌ	ラ	ぬ	ら
8	B_S	C_N	(	8	H	X	h	x	ぃ	く	ィ	ク	ネ	リ	ね	り
9	H_T	E_M	)	9	I	Y	i	y	ぅ	け	ゥ	ケ	ノ	ル	の	る
A	L_F	S_B	*	:	J	Z	j	z	ぇ	こ	ェ	コ	ハ	レ	は	れ
B	H_M	E_C	+	;	K	[	k	{	ぉ	さ	ォ	サ	ヒ	ロ	ひ	ろ
C	C_L	→	,	<	L	¥	l	|	ゃ	し	ャ	シ	フ	ワ	ふ	わ
D	C_R	←	−	=	M	]	m	}	ゅ	す	ュ	ス	ヘ	ン	へ	ん
E	S_O	↑	.	>	N	^	n	~	ょ	せ	ョ	セ	ホ	・	ほ	▦
F	S_I	↓	/	?	O	_	o	\	っ	そ	ッ	ソ	マ	°	ま	

4.2.2 In Noninterlace Mode



X / Y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		DE		0	@	P		p			〒	ー	タ	ミ		
1	SH	D1	!	1	A	Q	a	q			。	ア	チ	ム		円
2	SX	D2	"	2	B	R	b	r			「	イ	ツ	メ		年
3	EX	D3	#	3	C	S	c	s			」	ウ	テ	モ		月
4	ET	D4	$	4	D	T	d	t			、	エ	ト	ヤ		日
5	EQ	NK	%	5	E	U	e	u			・	オ	ナ	ユ		時
6	AK	SN	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		分
7	BL	EB	'	7	G	W	g	w			ァ	キ	ヌ	ラ		秒
8	BS	CN	(	8	H	X	h	x			ィ	ク	ネ	リ	♠	
9	HT	EM	)	9	I	Y	i	y			ゥ	ケ	ノ	ル	♥	
A	LF	SB	*	:	J	Z	j	z			ェ	コ	ハ	レ	◆	
B	HM	EC	+	;	K	[	k	{			ォ	サ	ヒ	ロ	♣	♪
C	CL	→	,	<	L	¥	l	|			ャ	シ	フ	ワ	●	÷
D	CR	←	−	=	M	]	m	}			ュ	ス	ヘ	ン	○	±
E	SO	↑	.	>	N	^	n	~			ョ	セ	ホ	゛		
F	SI	↓	/	?	O	—	o	\			ッ	ソ	マ	゜	\	

In both modes of interlace and noninterlace,
if X is 0 or 1, the code is **a control** code only.

4.3 ERROR MESSAGES OF LEVEL-3 DISK BASIC

Error message	Error code	Contents
Bad Data In File	58	Format of data on the file is incorrect.
Bad File Descriptor	55	An item not to be described was described in the file descriptor.
Bad File Mode	51	Execution of Input/Output instruction not corresponding to the mode of open time was attempted.
Bad File Number	50	The file of the designated number is not opened.
Bad File Structure	71	The file structure is erroneous (FAT or the directory is questionable.)
Bad Record Number	70	A record number of other than 1 through 624 was designated in PUT or GET.
Buffer Overflow	61	The I/O buffer overflowed. (The file cannot be read correctly, or others.)
Can't Continue	17	Cannot continue executing the program with the CONT command. (The pointer is broken.)
Device I/O Error	53	An error occurred in read or write of the device used.
Device In Use	59	Use of a busy device other than the disk was attempted (by a command or the OPEN instruction).
Device Unavailable	60	The device is not ready for use. (The interface is installed but not the connector.)
Direct Statement In File	56	A direct statement exists during load in the ASCII format (at the time of loading a data file).

- continued

Error message	Error code	Contents
Directory Full	64	No more files can be created since the directory is filled. (A directory for 40 files only is prepared for one diskette.)
Disk Full	66	Extension or creation of a file was attempted in the state that all groups have been used. * (No user area remains.)
Disk Write Protected	73	Write was attempted to a diskette with a Write-protect notch.
Division By Zero	11	0 was designated for the division divisor.
Drive Not Ready	72	No diskette is mounted to the designated drive.
Duplicate Definition	10	Double definition was attempted to an array or user function.
Field Overflow	68	Total of the size in the FIELD statement exceeded 128 bytes.
File Already Exists	67	A file name existing on the same diskette was designated for a new file in the NAME command.
File Already Open	52	Attempt to open an already opened file was made.
File Not Found	63	The designated file name does not exist on the disk.
File Not Open	57	Access to an unopened file was attempted. (Attempt was made to use an instruction that requires designation of a file number without opening the file.)
For Without Next	23	The FOR-NEXT is not in correct correspondence. (Too many FOR statements)

- continued

Error message	Error code	Contents
Illegal Direct	12	Execution of a command was attempted using a statement that cannot be used as a direct command.
Illegal Function Call	5	Naming of the statement function is in error.
Input Past End	54	A file input statement was executed after reading all data in the file.
Missing Operand	22	Necessary parameters in the statement are not designated.

* Execute the CLOSE instruction and press the RETURN key with the diskette being inserted in the drive when a Disk Full error occurs.

Error message	Error code	Contents
Next Without For	1	The FOR-NEXT is not in correct correspondence. (Either too many NEXT, or two or more FOR-NEXT loops are crossing.)
No Resume	19	Cannot restart on program execution after error handling.
Out Of Data	4	Data to be read by the READ statement is not prepared in the DATA statement. (Either there is no DATA statement or there is another READ statement after reading data.)
Out Of Memory	7	The memory capacity has been used up. (The program is too long, or the array is too large.)
Out Of String Space	14	Shortage of the memory area for the string variable designated in the CLEAR or other statement.
Overflow	6	The operation results or input numerics are outside of the allowed range.
Protected Program	62	Attempt was made to write or modify to a protected program.
Resume Without Error	20	The error and RESUME are not in correct correspondence. (RESUME was tried while there was no error.)
Resume Without Gosub	3	The GOSUB-RETURN is not in correct correspondence. (Only RETURN statement exists.)
String Formula Too Complex	16	The character expression is too complicated. (Nesting inside the parentheses is too big, and others.)

- continued

Error message	Error code	Contents
String Not Fielded	69	The character variable to be assigned by the LSET or RSET statement is not defined in the FIELD statement, or, change of the contents was attempted by an instruction of other than LSET or RSET to the character variable defined by the FIELD statement.
String Toc Long	15	The string (a string placed between two quotation marks) is too long. (Exceeded 255 characters.)
Subscript Out Of Range	9	The subscript of the array variable is not in the specified range.
Syntax Error	2	The program is not in the correct syntax. There is an illegal statement in the program.
Too Many Open Disk Files	65	Concurrent opening of many files exceeding the number of FCB (File Control Block) was attempted.
Type Mismatch	13	Data type of the right side and left side of the expression or argument of the function is not consistent (a numeric and string, or other discrepancy).
Undefined Line Number	8	There is no needed line in the program (destination for GOTO).
Undefined User Function	18	Referencing to a user function that was not defined in the DEFFN was attempted.
Unprintable Error	21 26~49 74~255	An error of which no message has been defined occurred.

- continued

Error message	Error code	Contents
WEND without WHILE	25	WHILE and WEND are not correctly corresponding.
WHILE without WEND	24	WHILE and WEND are not correctly corresponding.

FORMAT　　EOF(<file number>)

EXPLANATION　　EOF stands for End OF FIle, and it is the function to return whether or not a file in sequential file system reached the end as a value.

If the end is reached, the value is true (-1), and if not reached, the value is untrue (0). Therefore, a sentence like ⌜IF EOF (n) THEN⌟ is possible. (The sentence is the same as ⌜IF EOF (n)<> 0⌟ .)

2.3.16 ERR, ERL

PURPOSE　　To give an error number and error line number.

2.3.17 EXP

PURPOSE　　Exponent

2.3.18 FIX

PURPOSE　　To give an integer part.

2.3.19 FRE

PURPOSE　　TO give an unused area.

2.3.20 HEX$

PURPOSE　　Hexadecimal conversion

2.3.21 INKEY$

PURPOSE　　To give an input character input from the keyboard.

2.3.22 INPUT$ (Character input from input file)

PURPOSE Character input from the keyboard an input file

FORMAT INPUT$(<Number of characters>[,[#]<file number>])

EXPLANATION THis function reads in character stri of the designated number of character from the keyboard or from the designa input file. The <number of character is in the range of 0 through 255, and fraction part is rounded by counting fractions of .5 and over as a whole number and disregarding the rest. Th input characters are not displayed or screen. All characters input by keys other than the keys to stop the execu are input as is.

When the <file number> is omitted, ir is from the keyboard only.
When the <file number> is designated, pertinent file must be opened.

2.3.23 INSTR

PURPOSE Search of character string

2.3.24 INT

PURPOSE To make an integer

2.3.25 LEFT$

PURPOSE To give left part of a character str

2.3.26 LEN

PURPOSE To give a character string length.

2.3.27 LOC (Record No. for next access)

PURPOSE To give a record number that is GET or PUT after the random file.

FORMAT LOC(<file number>)

EXPLANATION LOC stands for Location Counter of disk file and this function gives the number next to the record that was put or get at the end of the random file.

The value is the same as default value when the record number is omitted in the PUT or GET Instruction.

Initial value of LOC is 1.

If the file of the <file number> is not opened in "R" mode (random access), it causes an error.

2.3.28 LOF (Maximum Record No. of random file)

PURPOSE To give the max. record number of the random file.

FORMAT LOF(<file number>)

EXPLANATION LOF stands for Length of disk File, and the function gives the maximum record number of the random file.

Beware that LOF does not change unless PUT is executed to a record having a greater record number than LOF.

If the file carrying the number of <file number> has not been opened in the "R" mode (random access), it causes an error.

2.3.9 LOG

PURPOSE — To give a natural logarithm.

2.3.30 MID$

PURPOSE — Extraction of a character string

2.3.31 MKI$, MKS$, MKD$ (Conversion to character data)

PURPOSE — Conversion of numerics to character type data

FORMAT

MKI$(<argument>)
MKS$(<argument>)
MKD$(<argument>)

EXPLANATION — MKI$, MKS$, MKD$ stands for Make Integer $ (or Single $ or Double $), and numerics are converted to character type data by these functions.

Data stored in random files must be in character type. Accordingly, numeric data must be converted to character type and these functions are used for the purpose. MKI$ converts an integer to a character string of 2 bytes, and MKS$ converts a single precision real number to a character string of 4 bytes, and MKD$ converts a double precision real number to a character string of 8 bytes. Since these functions convert numerics by recognizing internal expression of the numerics as character code, the number of bytes required is smaller than that required in conversion by STR$.

Data converted by MKI$, MKS$ and MKD$ can be recovered to the original numerics by the CVI, CVS, and CVD functions.

The MKI$, MKS$ and MKD$ functions can be used unrelated to file operation.

2.3.32 OCT$

PURPOSE Octal conversion

2.3.33 PEEK

PURPOSE Read of memory address.

2.3.34 PEN

PURPOSE To give light pen information.

2.3.35 POINT

PURPOSE To give information whether or not a point is present on the screen.

2.3.36 POS

PURPOSE To give the cursor horizontal position.

2.3.37 RIGHT$

PURPOSE To give right part of a character string.

2.3.38 RND

PURPOSE To give a random number.

2.3.39 SCREEN

PURPOSE TO give screen information.

2.3.40 SGN

PURPOSE To give a sign.

2.3.41 SIN

PURPOSE To give a sine.

2.3.42 SPACE$

PURPOSE To give a blank character string.

2.3.43 SPC

PURPOSE To output a blank

2.3.44 SQR

PURPOSE To give a square root.

2.3.45 STR$

PURPOSE Conversion of numeric to characters

2.3.46 STRING$

PURPOSE Conversion of character code to a character string

2.3.47 TAB

PURPOSE To move the cursor to the designated position.

2.3.48 TAN

PURPOSE To give a tangent.

2.3.49 TIME

PURPOSE — To give the seconds of the internal clock. (Assignment is not permitted.)

2.3.50 TIME$

PURPOSE — Time of the internal clock (Assignment is permitted.)

2.3.51 USR

PURPOSE — To call a machine language routine.

2.3.52 VAL

PURPOSE — Conversion of a character string to a numeric

2.3.53 VARPTR

PURPOSE — TO give a variable storing address.